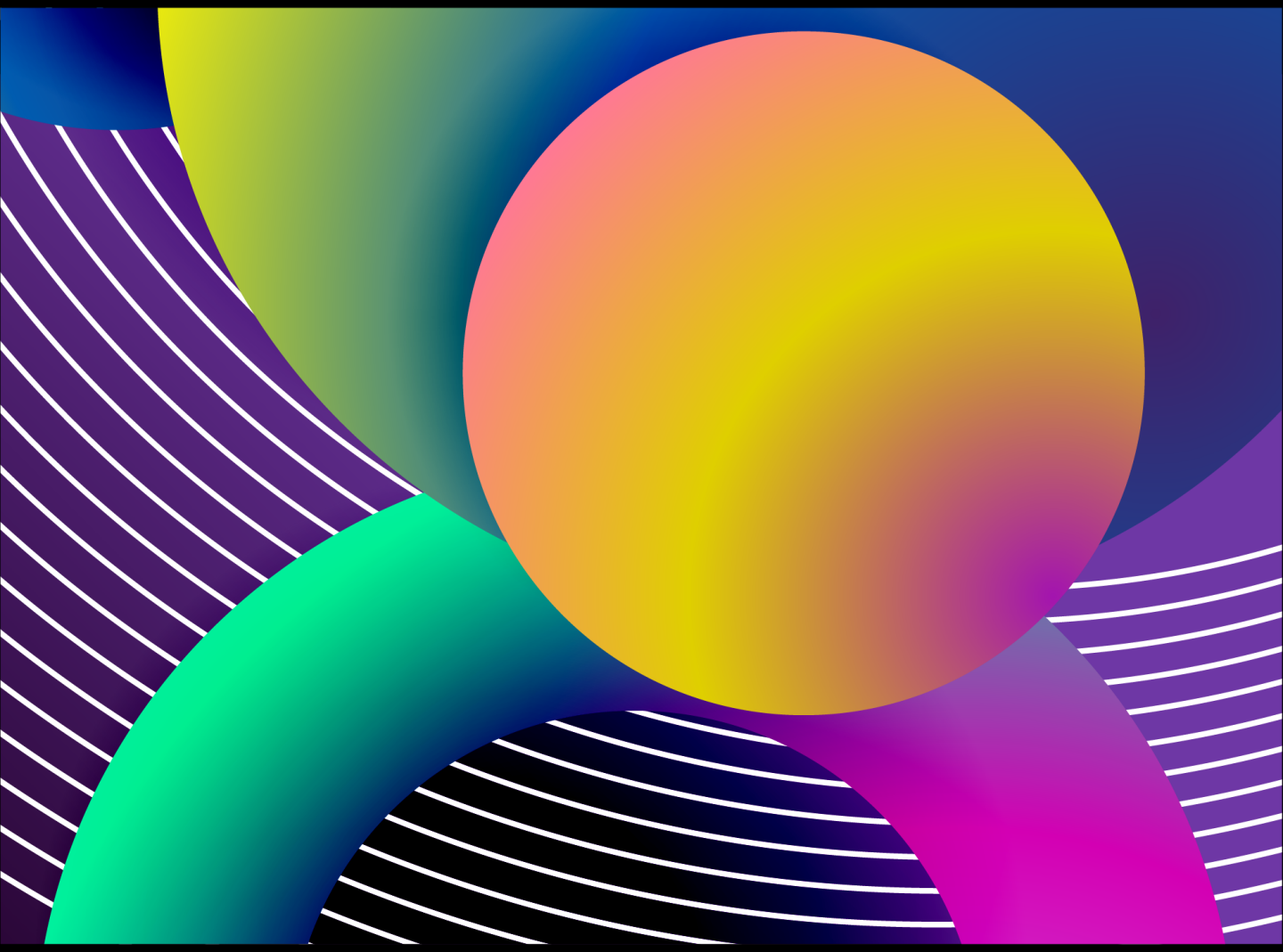
Jetpack Compose 1.8 Essentials



Payload publishing

Jetpack Compose 1.8 Essentials

Jetpack Compose 1.8 Essentials

ISBN-13: 978-1-965764-18-3

Rev: 1.0

https://www.payloadbooks.com

Chapter 41

41. Designing Adaptable List-Detail Layouts

In the previous chapter, we built a scrollable list containing items that displayed messages when clicked. A more common design paradigm when working with lists is the list-detail layout. In this chapter, we will explore how to build list-detail layouts that automatically adapt to device screen size and orientation. Once the basics have been covered, we will extend the LazyListDemo project by adding list-detail behavior.

Before we begin, we need to issue a brief disclaimer. At the time of writing, the NavigableListDetailPaneScaffold composable covered in this chapter was at the experimental development stage, and details may have changed in the meantime.

41.1 Introducing NavigableListDetailPaneScaffold

Android provides several ways to implement list-detail layouts, each offering varying levels of complexity and capabilities. A solution that combines extensive functionality with minimal coding effort is the NavigableListDetailPaneScaffold composable. This composable makes it easy to implement navigable layouts consisting of list and detail panes and supports an optional third pane (referred to as an extra pane).

NavigableListDetailPaneScaffold is fully adaptive, automatically arranging the pane layout relative to the available screen space. On a narrower screen, such as a phone in portrait orientation, the layout is presented as a series of individual screens (shown in Figure 39-1) through which the user navigates back and forth:

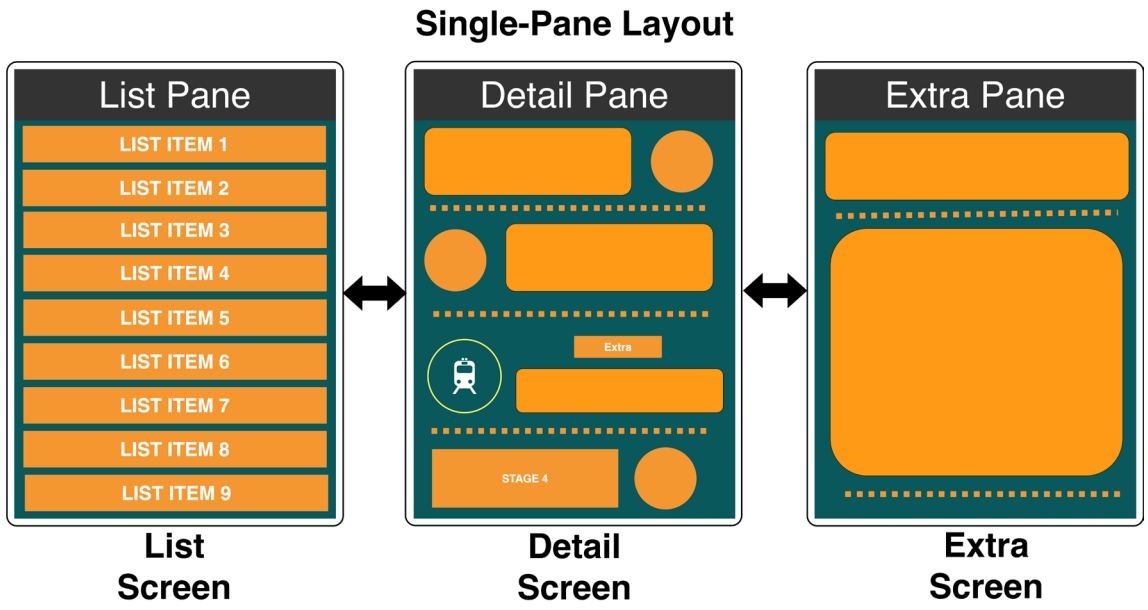


Figure 41-1

In wider configurations, such as a tablet device in landscape orientation, the scaffold switches to a multi-pane

layout, where panes are displayed side by side. Each pane appears adjacent to the current pane in response to a navigation selection and disappears as the user navigates backward. Figure 39-2 illustrates a multi-pane layout with the list and detail panes visible:

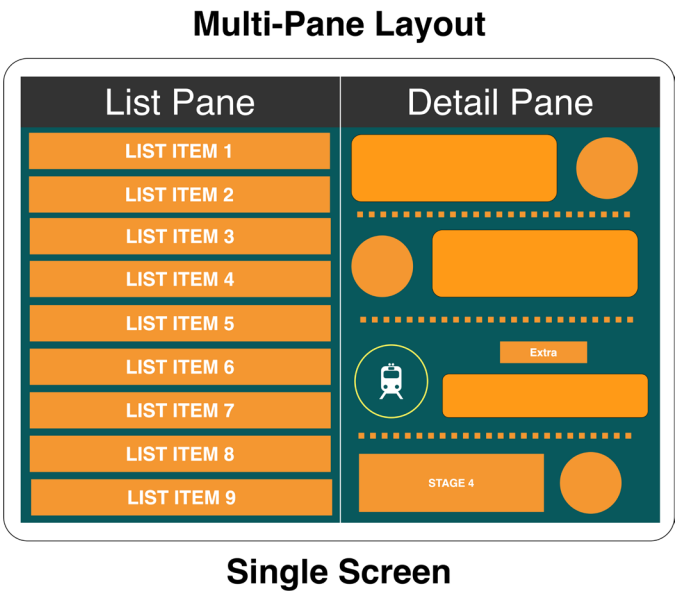


Figure 41-2

The scaffold will only display two panes at a time, so if the extra pane is displayed, the list pane will be hidden.

In its simplest form, a list-detail implementation consists of a scaffold navigator, code to render the list and detail panes, and, if required, the optional extra pane. To add transition animation, these panes can be embedded within AnimatedPane instances. The navigator and panes are then passed to a NavigableListDetailPaneScaffold instance for presentation.

41.2 The scaffold navigator

A scaffold navigator is an instance of the ThreePaneScaffoldNavigator interface that manages and controls navigation between panes. It is also responsible for saving and restoring the currently selected list item during app invocations and configuration changes, such as rotating the device from portrait to landscape orientation.

Scaffold navigator instances are created by calling the *rememberListDetailPaneScaffoldNavigator()* function, which you provide a custom class with key-value properties that are suitable for saving and restoring the current navigation state. This is accomplished by declaring the custom class as parcelable, which allows class instances to be saved and restored with the same property values.

The following code declares a parcelable class and demonstrates how to use it to create a ThreePaneScaffoldNavigator instance:

```
@Parcelize
class ListItem(val id: Int) : Parcelable

val scaffoldNavigator = rememberListDetailPaneScaffoldNavigator<ListItem>()
```

41.3 NavigableListDetailPaneScaffold syntax

Once we have an initialized scaffold navigator, the next step is to implement the NavigableListDetailPaneScaffold composable using the following syntax:

```
NavigableListDetailPaneScaffold(
    navigator = <scaffold navigator>,
    listPane = {
        // Code for list pane here
    },
    detailPane = {
        // Code for detail pane here
    },
    extraPane = {                   // <- Optional
        // Code for extra pane here
    }
    defaultBackBehavior: <BackNavigationBehavior>,
    paneExpansionDragHandle: @Composable (ThreePaneScaffoldScope
              .(PaneExpansionState) -> Unit)? = null,
    paneExpansionState: PaneExpansionState? = null
)
```

NavigableListDetailPaneScaffold is a slot API composable, and composables for the listPane and detailPane slots are mandatory. In addition, a scaffold navigator must be assigned to the navigator property.

The *defaultBackBehavior* property defines what happens when the user navigates backward within the list-detail layout, and the pane expansion parameters allow the addition of a drag handle to adjust the size of the visible panes in multi-pane layouts.

41.4 Navigating between panes

Once the navigation scaffold is initialized and visible, it needs to know when and how to navigate between the content panes. We need to tell it, for example, to navigate to the detail pane when an item in the list pane is selected. Navigation is achieved by calling the *navigateTo()* method of the scaffold navigator object, passing it a property from the ListDetailPaneScaffoldRole object corresponding to the destination pane (List, Detail, or Extra) and the selected item. *navigateTo()* is a suspend function, so it must be called from within a coroutine.

The code to navigate to the detail pane when a list item is clicked might, therefore, read as follows:

```
coroutineScope.launch {
    scaffoldNavigator.navigateTo(
        ListDetailPaneScaffoldRole.Detail,
        item
    )
}
```

When the destination pane is invoked, the selected item can be identified by accessing the scaffold navigator's *currentDestination* property:

```
val currentDestination = scaffoldNavigator.currentDestination
```

The current destination is an instance of the ThreePaneScaffoldDestinationItem class from which we can extract the key of the current item and pass it to the destination pane:

```
val currentContentKey = currentDestination?.contentKey
MyDetailPane(currentContentKey)
```

Since the *currentDestination* and *contentKey* properties are nullable types, the above steps are best performed within a *let* statement, as outlined below:

```
scaffoldNavigator.currentDestination?.contentKey?.let {
    MyDetailPane(it)
}
```

The remainder of this chapter will put theory into practice by adding adaptable list-detail layout support to the LazyListDemo project.

41.5 Preparing the LazyListDemo project

Launch Android Studio and open the LazyListDemo project created in the previous chapter. If you are not reading this book sequentially and want to skip the steps in the previous chapter, download the project source code from the following URL and use Android Studio to navigate to and open the LazyListDemo folder:

https://www.payloadbooks.com/product/compose18/

Once the project is open, edit the *Gradle Scripts -> libs.versions.toml* file and modify it to add the adaptive navigation module to the build configuration:

```
[versions]
.
.
[libraries]
androidx-adaptive-navigation = { module = "androidx.compose.material3.
adaptive:adaptive-navigation" }
.
.
```

Insert the matching dependency shown below into the module-level *build.gradle.kts* file. The *kotlin-parcelize* plugin will also be required when we declare the item class:

```
plugins {
    alias(libs.plugins.android.application)
    alias(libs.plugins.kotlin.android)
    alias(libs.plugins.kotlin.compose)
    id("kotlin-parcelize")
}
.
.
dependencies {
   implementation(libs.androidx.adaptive.navigation)
.
.
}
```

Click the *Sync Now* link to commit the changes to the build configuration.

41.6 Designing the detail and extra panes

The project already includes a list pane, so only the detail and extra panes need to be added. The detail pane consists of a Card composable containing a text title, an image, and a button to navigate to the extra pane. The detail function will accept the selected CarItem object and the list of car models as parameters. The extra pane will consist of a Box layout containing a Text component.

Edit the *MainActivity.kt* file and declare the CarItem class:

```
.
.
import android.os.Parcelable
import kotlinx.parcelize.Parcelize

@Parcelize
class CarItem(val id: Int) : Parcelable

@OptIn(ExperimentalMaterial3AdaptiveApi::class)
class MainActivity : ComponentActivity() {
.
.
```

Next, add the DetailPane and ExtraPane composable functions:

```
.
.
import androidx.compose.foundation.background
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.material3.Button
import androidx.compose.runtime.rememberCoroutineScope
import androidx.compose.material3.adaptive.ExperimentalMaterial3AdaptiveApi
import androidx.compose.material3.adaptive.navigation.ThreePaneScaffoldNavigator
import kotlinx.coroutines.launch
.
.
@OptIn(ExperimentalMaterial3AdaptiveApi::class)
@Composable
fun DetailPane(
    item: CarItem,
    carList: List<String>,
    scaffoldNavigator: ThreePaneScaffoldNavigator<CarItem>,
    modifier: Modifier = Modifier
) {
    val model = carList[item.id]
    val scope = rememberCoroutineScope()

    Card(
        colors = CardDefaults.cardColors(
```

```
            containerColor = MaterialTheme.colorScheme.surfaceContainer
        ),
    ) {
        Column(
            modifier = modifier.fillMaxSize(),
            horizontalAlignment = Alignment.CenterHorizontally
        ) {
            Spacer(Modifier.size(32.dp))
            Text(
                text = model,
                style = MaterialTheme.typography.headlineLarge,
            )
            ImageLoader(model, modifier = Modifier.fillMaxSize(0.9f))

            Button(onClick = {
                scope.launch {
                    scaffoldNavigator.navigateTo(
                        ListDetailPaneScaffoldRole.Extra,
                        item
                    )
                }
            }) {
                Text(text = "Extra")
            }
        }
    }
}

@Composable
fun ExtraPane(modifier: Modifier = Modifier) {
    Box(
        modifier = modifier
            .fillMaxSize()
            .background(MaterialTheme.colorScheme.tertiaryContainer),
        contentAlignment = Alignment.Center
    ) {
        Text(
            text = "Extra Pane",
            style = MaterialTheme.typography.headlineLarge
        )
    }
}
```

41.7 Creating the scaffold navigator

With the new content panes designed, the next requirement is the scaffold navigator. Remaining in the *MainActivity.kt* file, locate the MainScreen function and modify it to create the navigator:

```
.
.
import androidx.compose.material3.adaptive.navigation.rememberListDetailPaneScaff
oldNavigator
.
.
@OptIn(ExperimentalMaterial3AdaptiveApi::class)
@Composable
fun MainScreen(carList: List<String>, modifier: Modifier = Modifier) {

    val scaffoldNavigator = rememberListDetailPaneScaffoldNavigator<CarItem>()
.
.
```

41.8 Modifying the ListPane and CarListItem functions

The original ListPane composable required only the car list and did nothing more than display a toast message in response to item clicks. Now that we are adding list-detail support, ListPane will need access to the scaffold navigator so that the toast message can be replaced with navigation to the detail pane. Locate the ListPane function and make the following changes:

```
.
.
import androidx.compose.material3.adaptive.layout.ListDetailPaneScaffoldRole
.
.
@OptIn(ExperimentalMaterial3AdaptiveApi::class)
@Composable
fun ListPane(
    modifier: Modifier = Modifier,
    carList: List<String>,
    scaffoldNavigator: ThreePaneScaffoldNavigator<CarItem>
) {
    val context = LocalContext.current
    val scope = rememberCoroutineScope()

    val onListItemClick = { text : String ->
_
        Toast.makeText(
            context,
            text,
            Toast.LENGTH_SHORT
        ).show()
    }

    LazyColumn(modifier) {
       carList.forEachIndexed { id, model ->
            item {
```