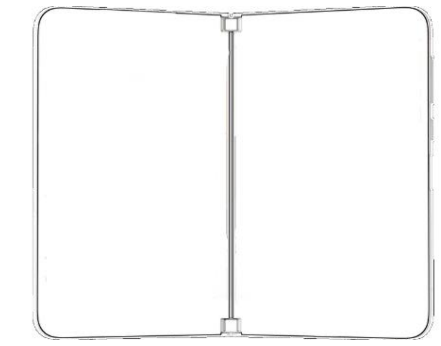
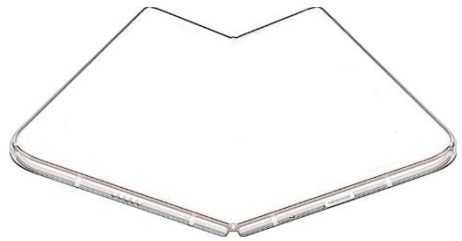
Different screen sizes

~~Foldables~~ on Android

What's new?

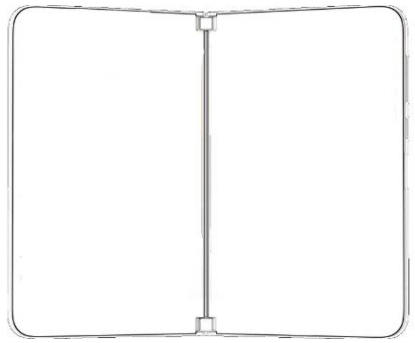
Thomas Künneth





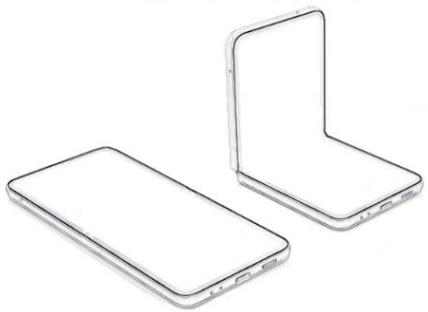
- What are foldables anyway?
- Window size classes and why we need them
- Canonical layouts
- Material3 Adaptive

https://github.com/tkuenneth/foldables_and_large_screens



- Book-like experience
- Typically, 360-degree hinge
- Various postures depending on intended use

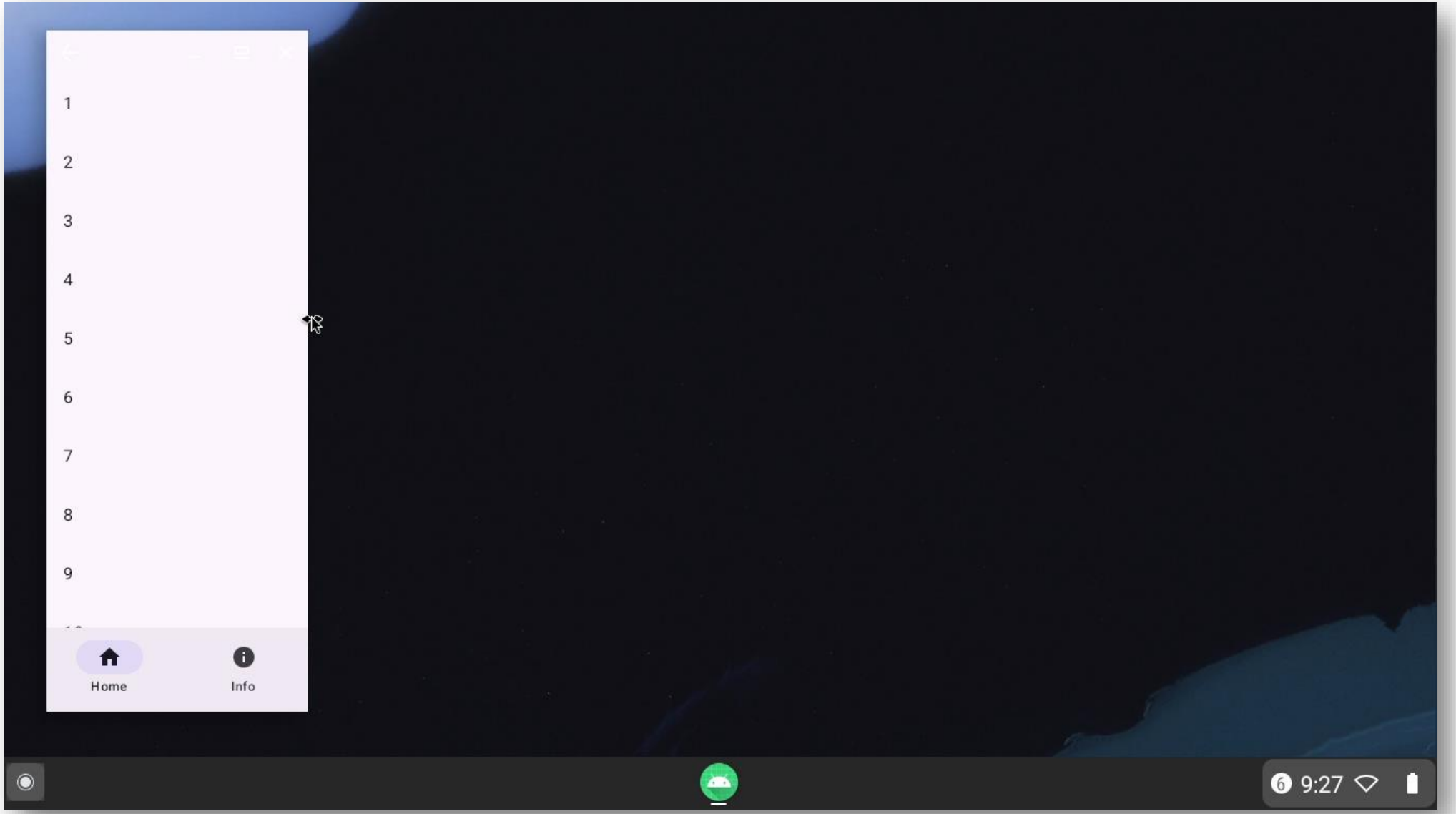
- Feels like a smartphone when closed
- Essentially becomes a tablet when opened
- Various postures, but no 360 degrees

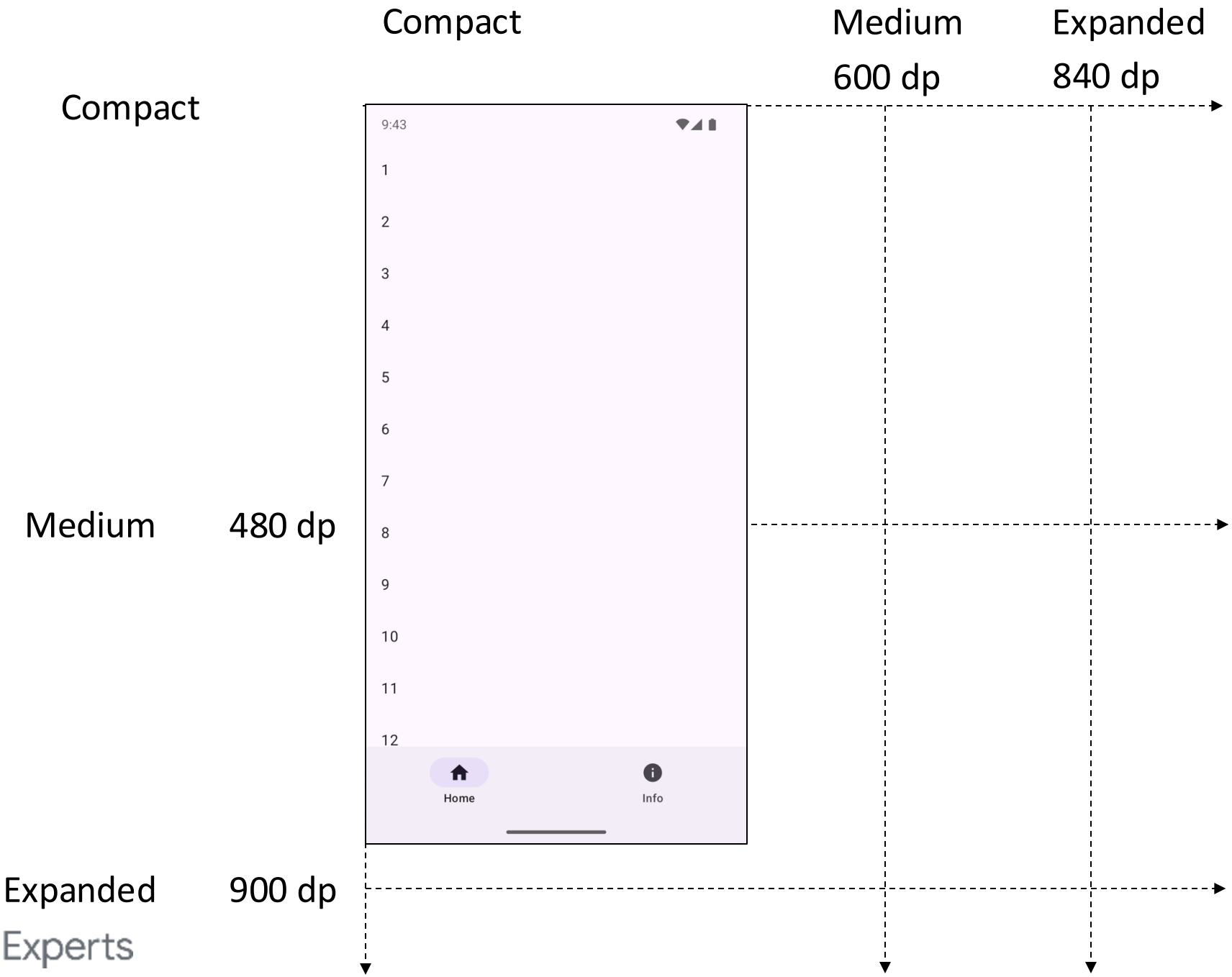


- Very compact
- Must be opened for full usage
- Main screen essentially is Smartphone-sized
- A small screen for ad hoc interactions

Not all foldables have large screens

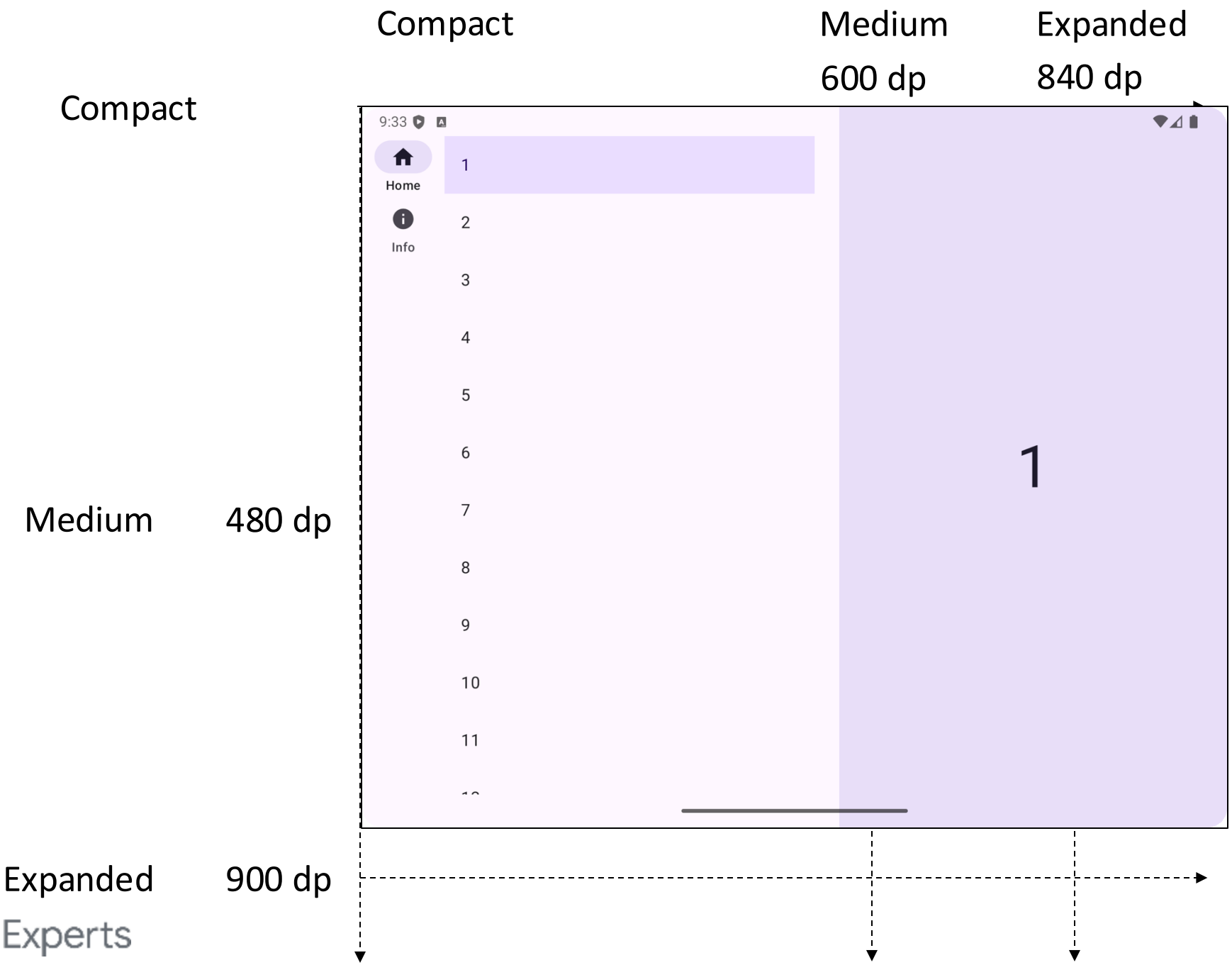
- Large screen
- Natural orientation portrait or landscape
- May offer a Freeform experience


1
2
3
4
5
6
7
8
9
Home
Info
9:27



Experts

SnappMobile



Experts

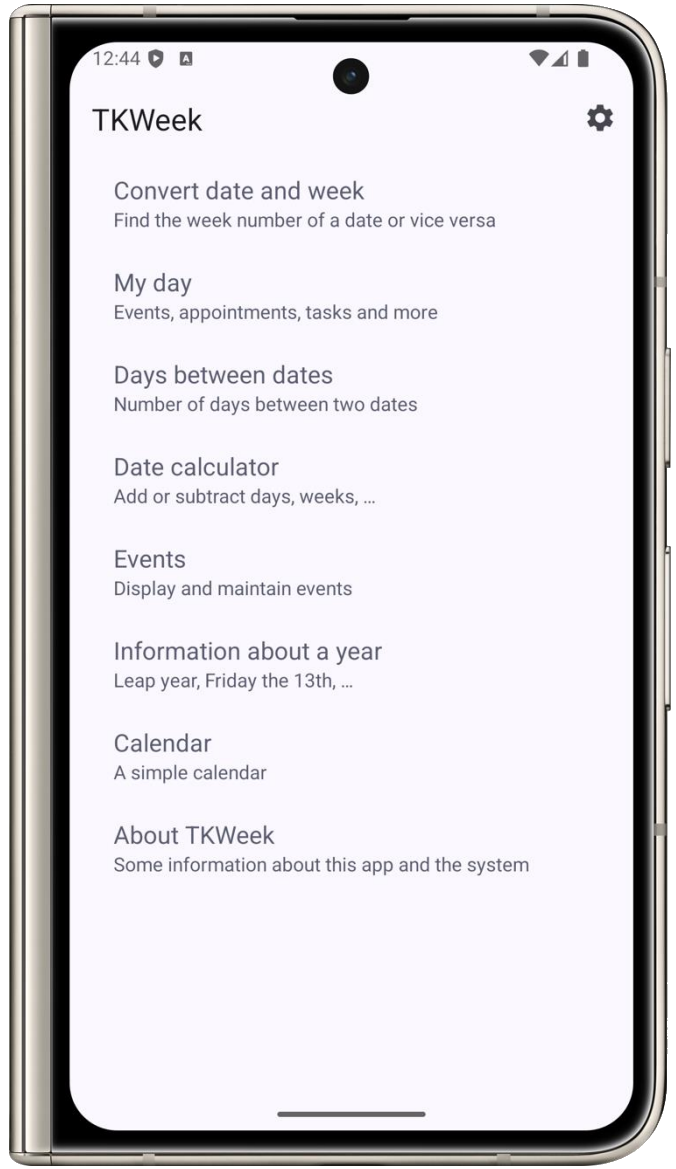
SnappMobile

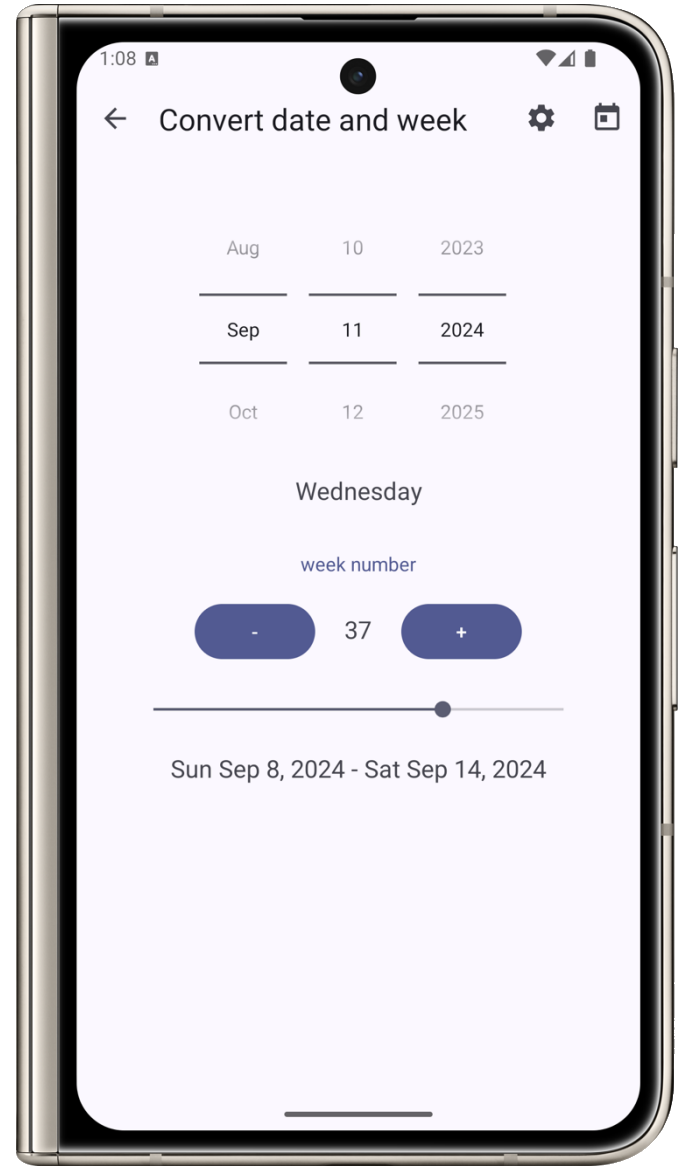
Window size classes affect the **location and size** of both **navigation** and **content area**, as well as **which components we use**

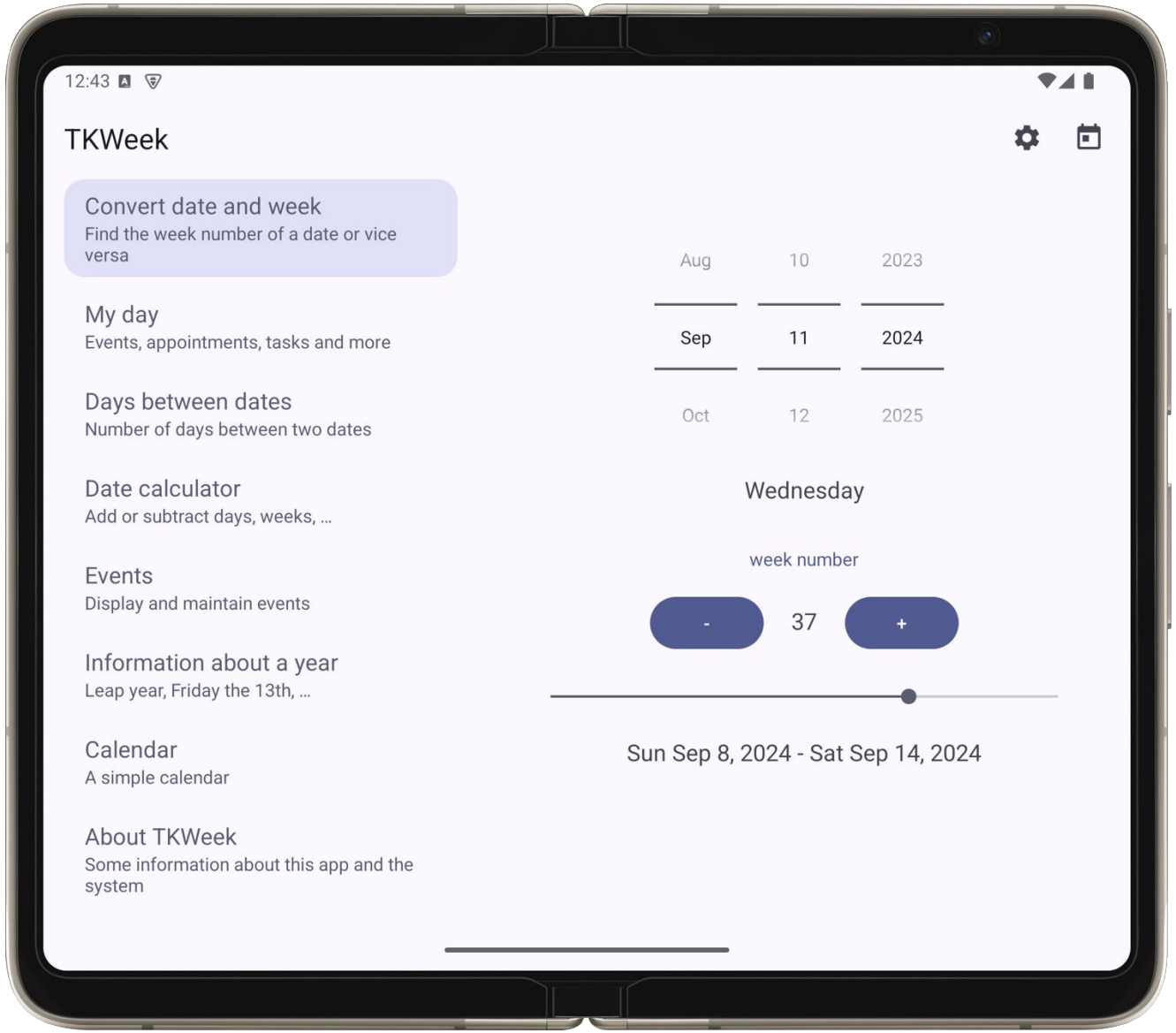
- Concept of window size classes introduced in 2021
- Two implementations (Jetpack WindowManager and material3-window-size-class)
- Until recently, logic which navigation composable to be used needed to be done in the app code

- **Navigation bar** for compact horizontal window size class
- **Navigation rail** for medium or expanded horizontal window size class
- **Navigation drawer** for expanded horizontal window size class
- Considerable amount of boilerplate code

What’s inside the content area?


12:44
TKWeek
Convert date and week
Find the week number of a date or vice versa
My day
Events, appointments, tasks and more
Days between dates
Number of days between two dates
Date calculator
Add or subtract days, weeks, …
Events
Display and maintain events
Information about a year
Leap year, Friday the 13th, …
Calendar
A simple calendar
About TKWeek
Some information about this app and the system


1:08
Convert date and week
Aug 10 2023
Sep 11 2024
Oct 12 2025
Wednesday
week number
- 37 +
Sun Sep 8, 2024 - Sat Sep 14, 2024

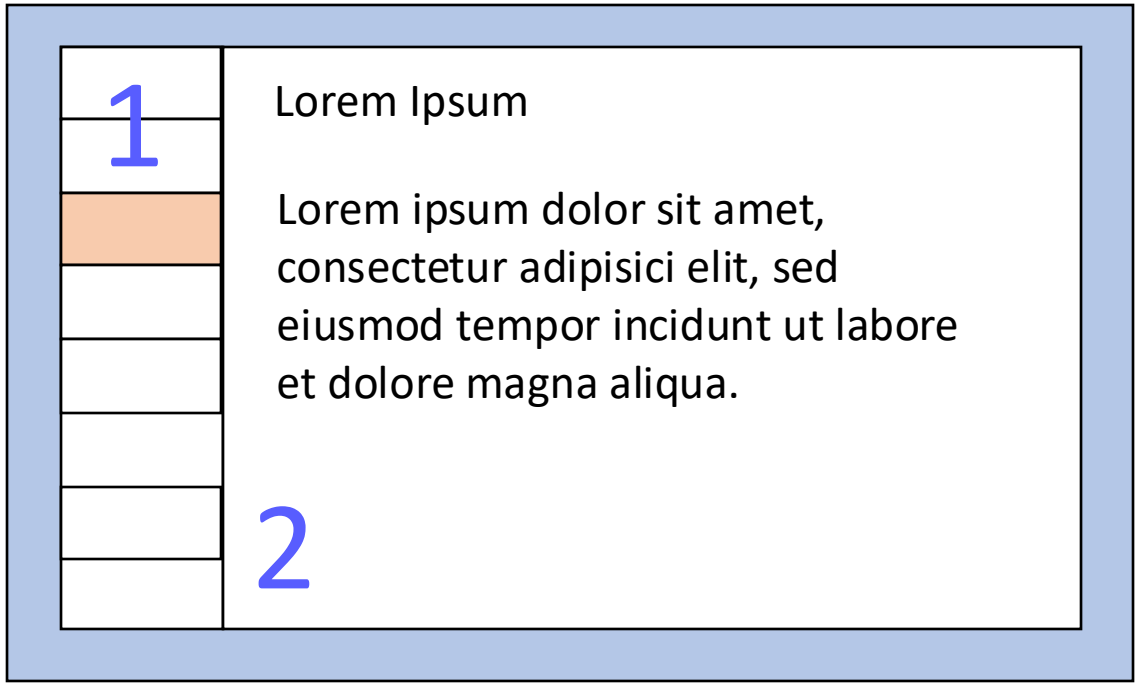


Canonical layouts

- Template for a specific type of layout or design
- Based on logical panes
- Implementations use Window Size Classes to determine …
  - which panes are visible
  - size, location, and content of panes
- Until recently, no ready-to-use Jetpack Compose implementations

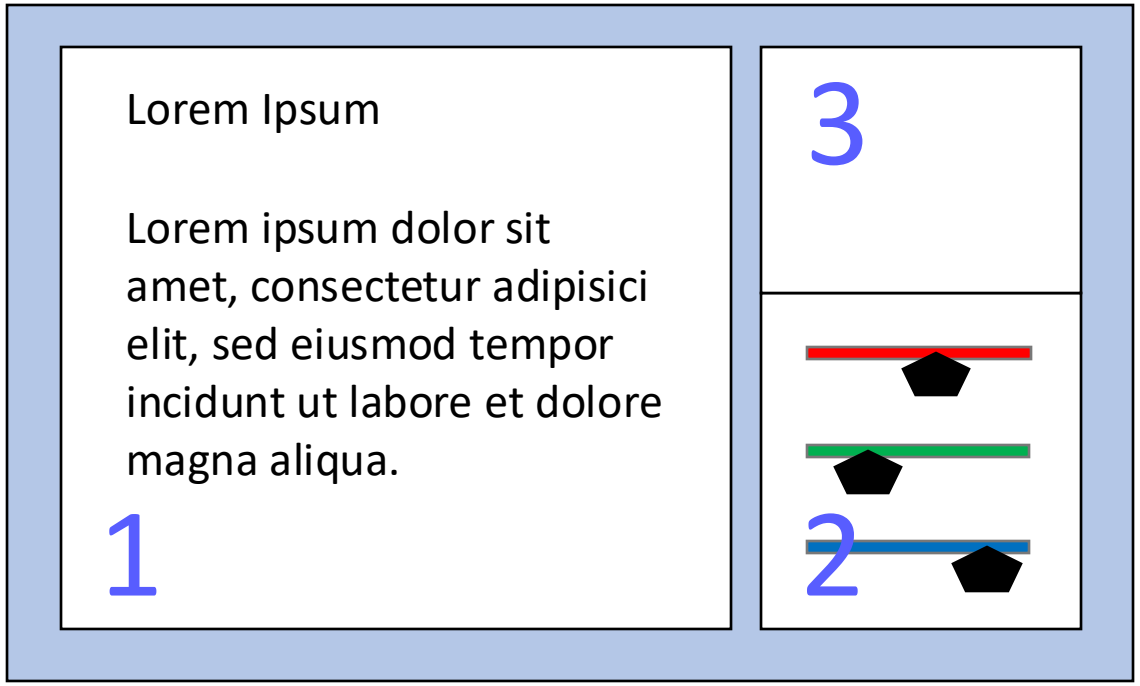
List-detail

- Scrollable lists of items
- Item detail containing supplementary information



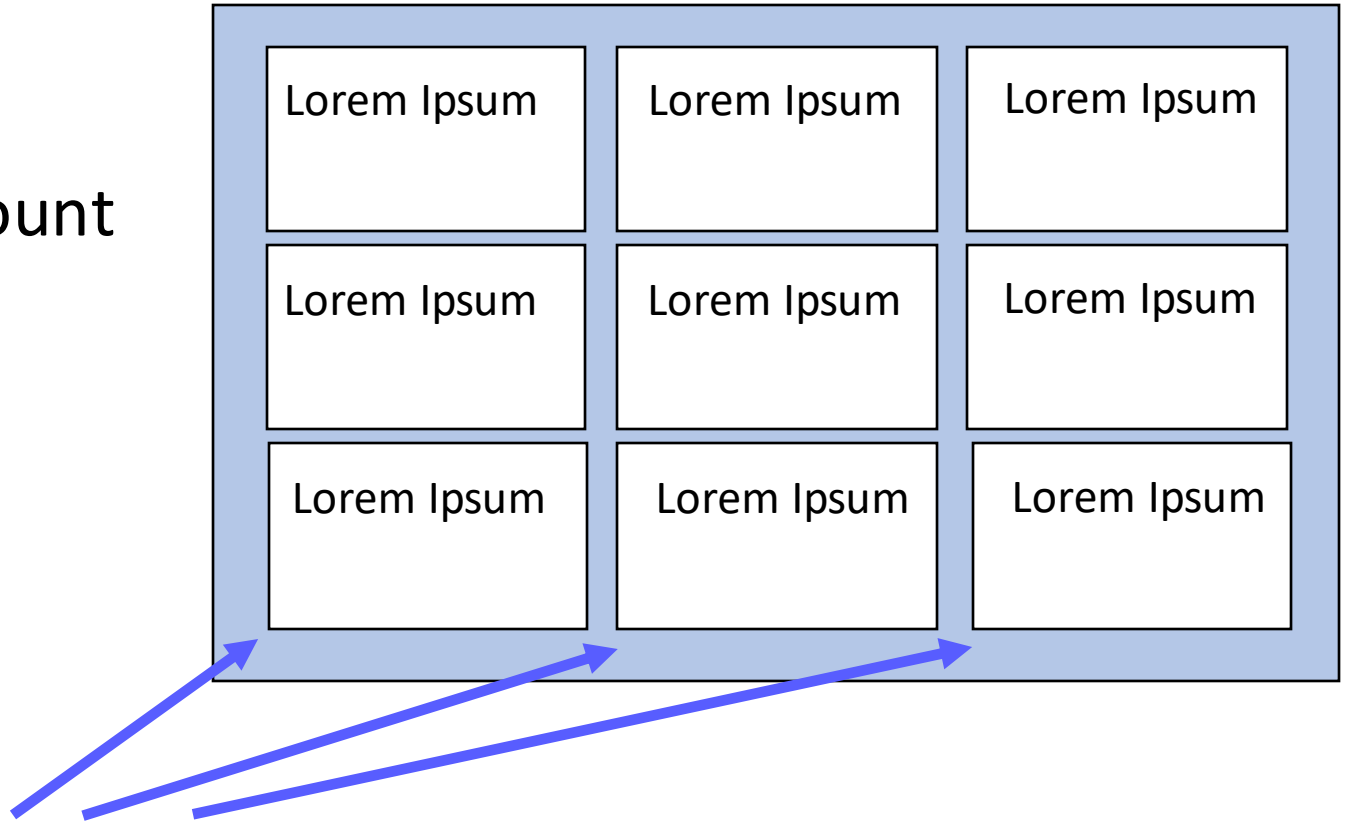
Supporting pane

- Primary section for the main app content
- Secondary section supports the main app content
- Optional tertiary section for additional content



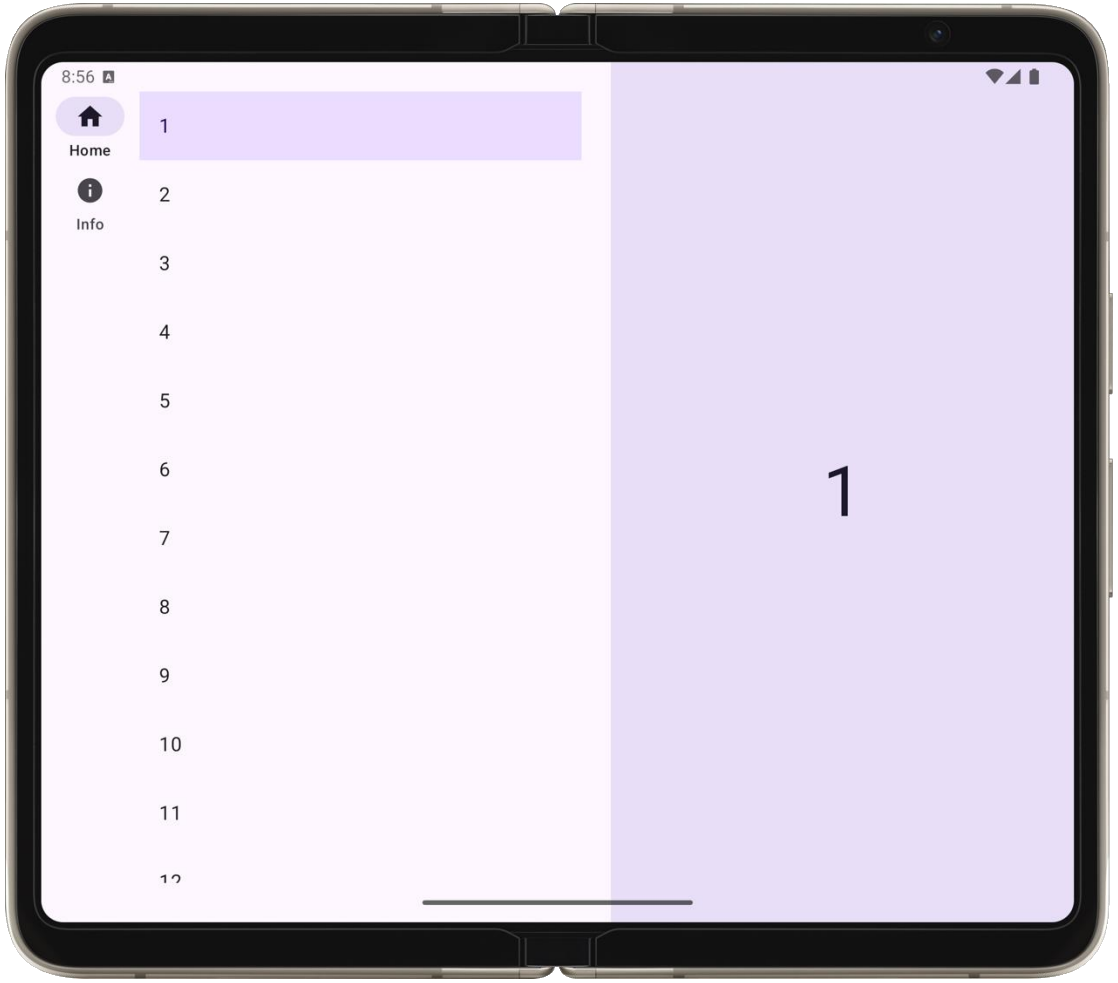
Feed

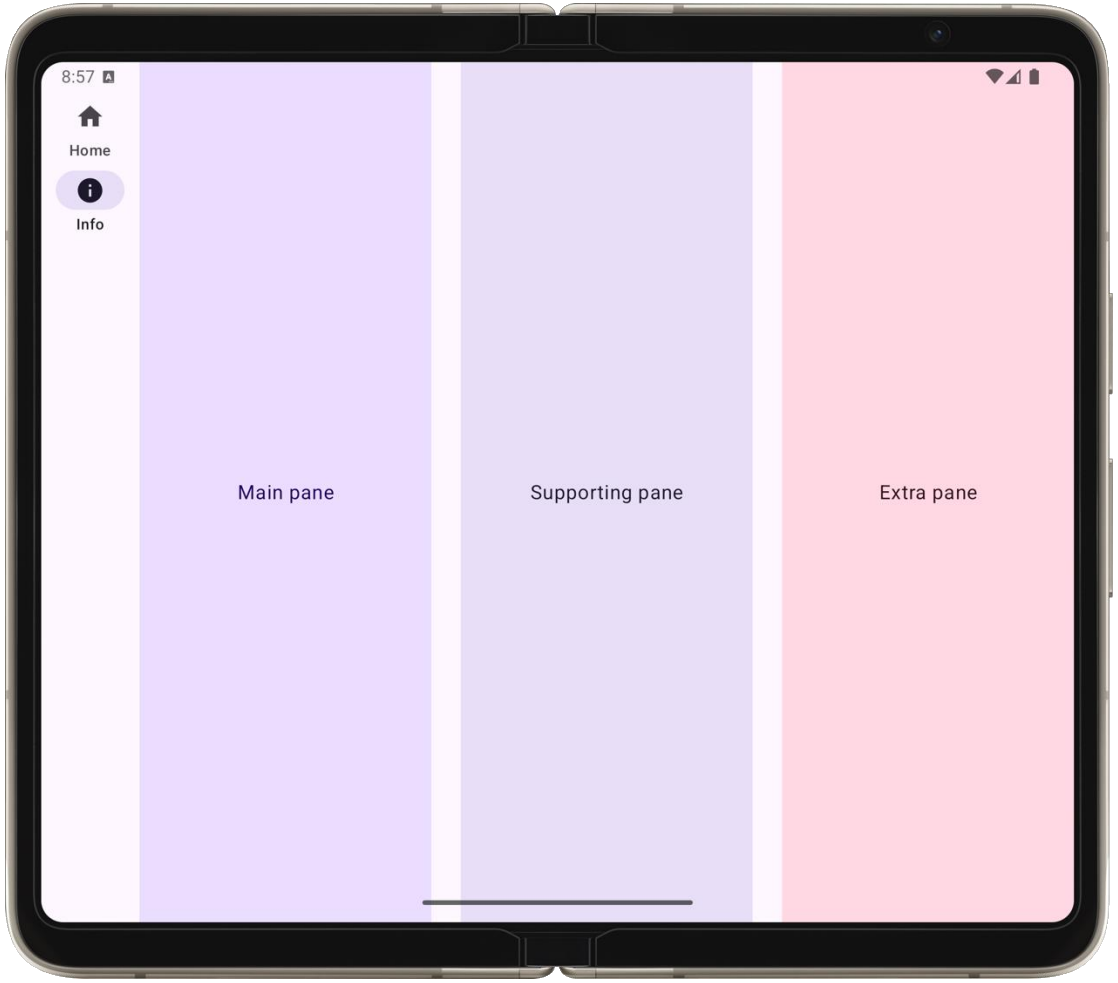
- Organizes cards in a grid
- Easily browse a large amount of content



Material3 Adaptive

- Uses navigation components based on window size classes
- Provides implementations of List-detail and Supporting pane
- Relies on panes


8:56
Home
Info
1
2
3
4
5
6
7
8
9
10
11
1


8:57
Home
Info
Main pane
Supporting pane
Extra pane

```
[versions]
material3AdaptiveNavSuite = "1.3.0"
material3Adaptive = "1.0.0"

[libraries]
androidx-material3-adaptive-navigation-suite = {
    module = "androidx.compose.material3:material3-adaptive-navigation-suite",
    version.ref = "material3AdaptiveNavSuite" }
androidx-material3-adaptive-layout = {
    module = "androidx.compose.material3.adaptive:adaptive-layout",
    version.ref = "material3Adaptive" }
androidx-material3-adaptive-navigation = {
    module = "androidx.compose.material3.adaptive:adaptive-navigation",
    version.ref = "material3Adaptive" }
```

```
dependencies {
    implementation(libs.androidx.material3.adaptive.navigation.suite)
    implementation(libs.androidx.material3.adaptive.layout)
    implementation(libs.androidx.material3.adaptive.navigation)
}
```

```kotlin
class MaterialAdaptiveDemoActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            MaterialTheme(colorScheme = defaultColorScheme()) {
                MaterialAdaptiveDemo()
            }
        }
    }
}
```

```kotlin
@Composable
fun MaterialAdaptiveDemo() {
    var currentDestination by rememberSaveable { mutableStateOf(AppDestinations.Home) }
    NavigationSuiteScaffold(navigationSuiteItems = {
        AppDestinations.entries.forEach {
            item(
                selected = it == currentDestination,
                onClick = { currentDestination = it },
                icon = {
                    Icon(
                        imageVector = it.icon,
                        contentDescription = stringResource(it.contentDescription)
                    )
                },
                label = { Text(text = stringResource(it.labelRes)) },
            )
        }
    }) {
        when (currentDestination) {
            AppDestinations.Home -> { HomePane() }

            AppDestinations.Info -> { InfoPane() }
}}}
```

```kotlin
enum class AppDestinations(
    @StringRes val labelRes: Int,
    val icon: ImageVector,
    @StringRes val contentDescription: Int = labelRes,
) {
    Home(
        labelRes = R.string.tab_home, icon = Icons.Default.Home
    ),
    Info(
        labelRes = R.string.tab_info, icon = Icons.Default.Info
    ),
}
```

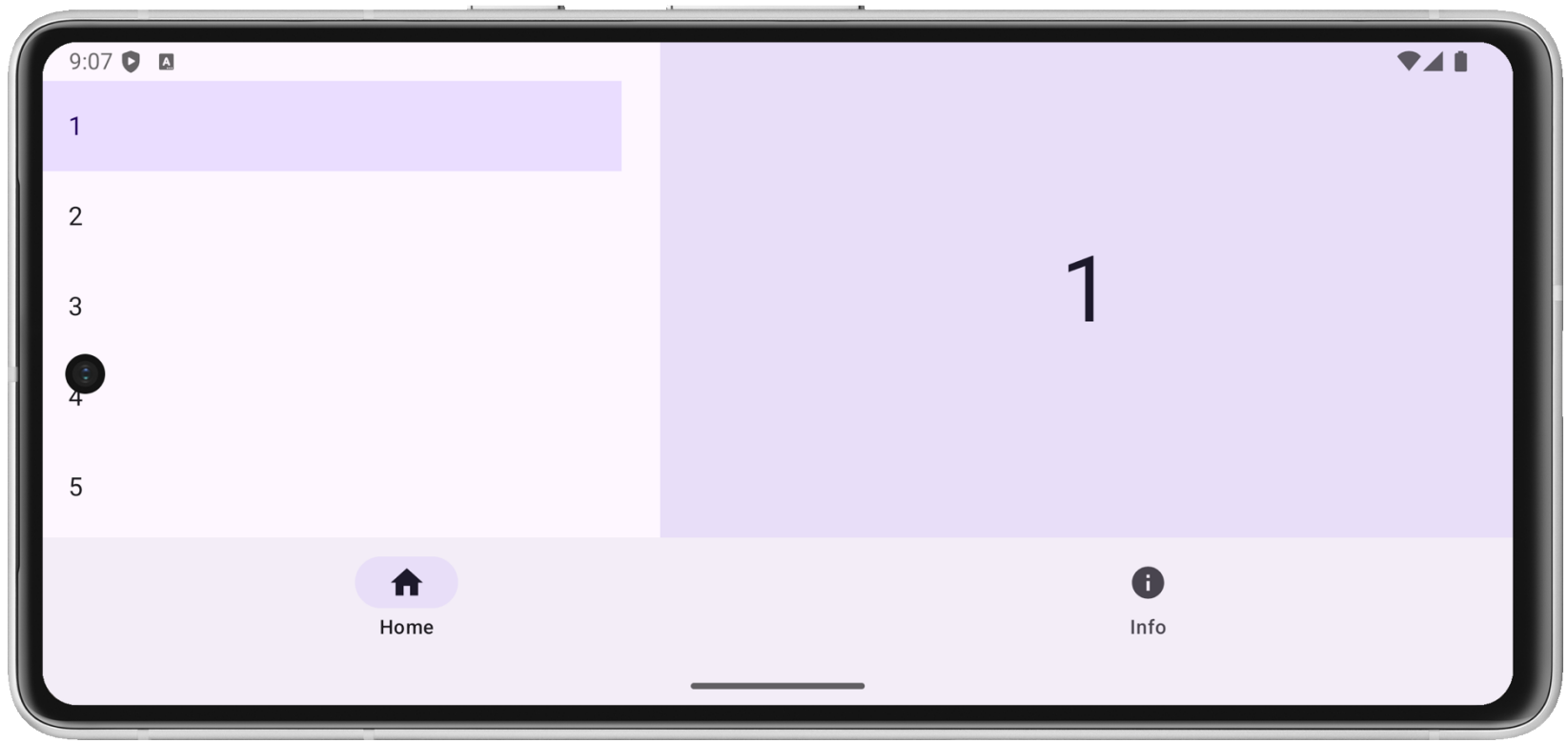
```kotlin
@Composable
fun MaterialAdaptiveDemo() {
    var currentDestination by rememberSaveable { mutableStateOf(AppDestinations.Home) }
    NavigationSuiteScaffold(navigationSuiteItems = {
        AppDestinations.entries.forEach {
            item(
                selected = it == currentDestination,
                onClick = { currentDestination = it },
                icon = {
                    Icon(
                        imageVector = it.icon,
                        contentDescription = stringResource(it.contentDescription)
                    )
                },
                label = { Text(text = stringResource(it.labelRes)) },
            )
        }
    }) {
        when (currentDestination) {
            AppDestinations.Home -> { HomePane() }

            AppDestinations.Info -> { InfoPane() }
}}}
```

```
sealed interface NavigationSuiteScope {
```

> This function sets the parameters of the default Material navigation item to be used with the Navigation Suite Scaffold. The item is called in `NavigationSuite`, according to the current `NavigationSuiteType`.
>
> For specifics about each item component, see `NavigationBarItem`, `NavigationRailItem`, and `NavigationDrawerItem`.
>
> Params: `selected` - whether this item is selected
> `onClick` - called when this item is clicked
> `icon` - icon for this item, typically an `Icon`
> `modifier` - the `Modifier` to be applied to this item
> `enabled` - controls the enabled state of this item. When `false`, this component will not respond to user input, and it will appear visually disabled and disabled to accessibility services. Note: as of now, for `NavigationDrawerItem`, this is always `true`.
> `label` - the text label for this item
> `alwaysShowLabel` - whether to always show the label for this item. If `false`, the label will only be shown when this item is selected. Note: for `NavigationDrawerItem` this is always `true`
> `badge` - optional badge to show on this item
> `colors` - `NavigationSuiteItemColors` that will be used to resolve the colors used for this item in different states. If null, `NavigationSuiteDefaults.itemColors` will be used.
> `interactionSource` - an optional hoisted `MutableInteractionSource` for observing and emitting `Interaction`s for this item. You can use this to change the item's appearance or preview the item in different states. Note that if `null` is provided, interactions will still happen internally.

```
    fun item(
```

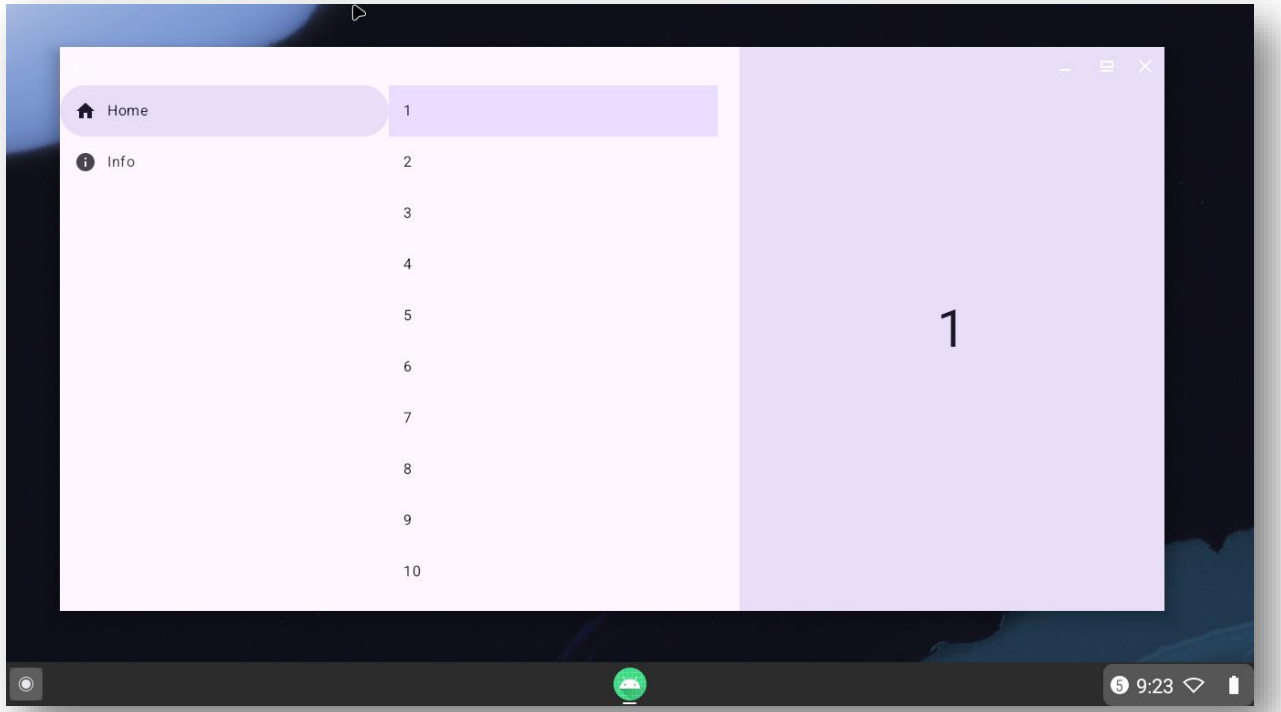
```
@Composable
fun MaterialAdaptiveDemo() {
    var currentDestination by rememberSaveable { mutableStateOf(AppDestinations.Home) }
    NavigationSuiteScaffold(navigationSuiteItems = {
        AppDestinations.entries.forEach {
            item(
                selected = it == currentDestination,
                onClick = { currentDestination = it },
                icon = {
                    Icon(
                        imageVector = it.icon,
                        contentDescription = stringResource(it.contentDescription)
                    )
                },
                label = { Text(text = stringResource(it.labelRes)) },
            )
        }
    }) {
        when (currentDestination) {
            AppDestinations.Home -> { HomePane() }

            AppDestinations.Info -> { InfoPane() }
}}}
```



Experts

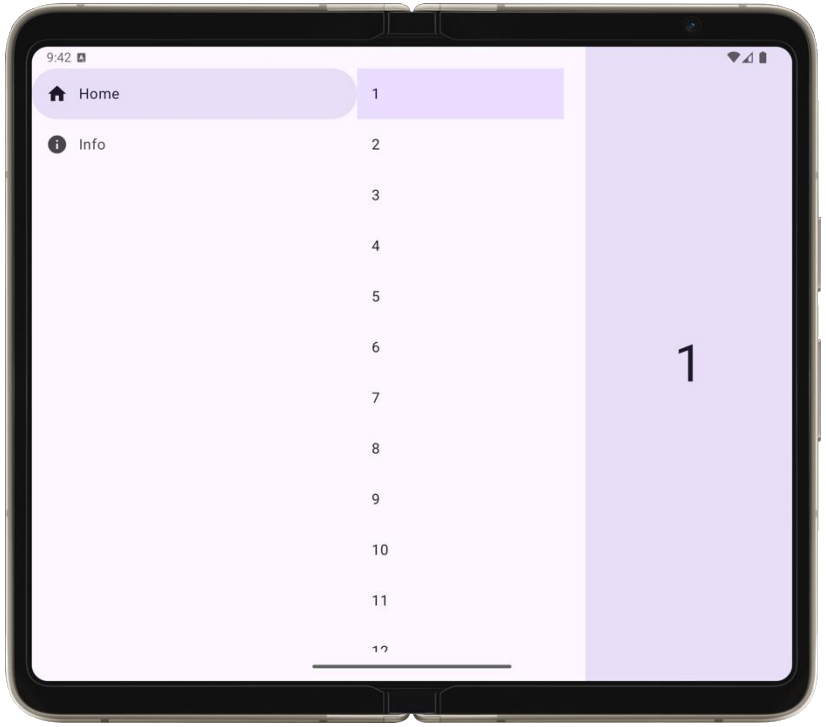
SnappMobile


```kotlin
val adaptiveInfo = currentWindowAdaptiveInfo()
val customNavSuiteType = with(adaptiveInfo.windowSizeClass) {
  if (windowWidthSizeClass != WindowWidthSizeClass.COMPACT &&
      windowHeightSizeClass == WindowHeightSizeClass.COMPACT) {
    NavigationSuiteType.NavigationRail
  } else {
    NavigationSuiteScaffoldDefaults.calculateFromAdaptiveInfo(adaptiveInfo)
  }
}
NavigationSuiteScaffold(
  layoutType = customNavSuiteType, navigationSuiteItems = { ... }
)
```



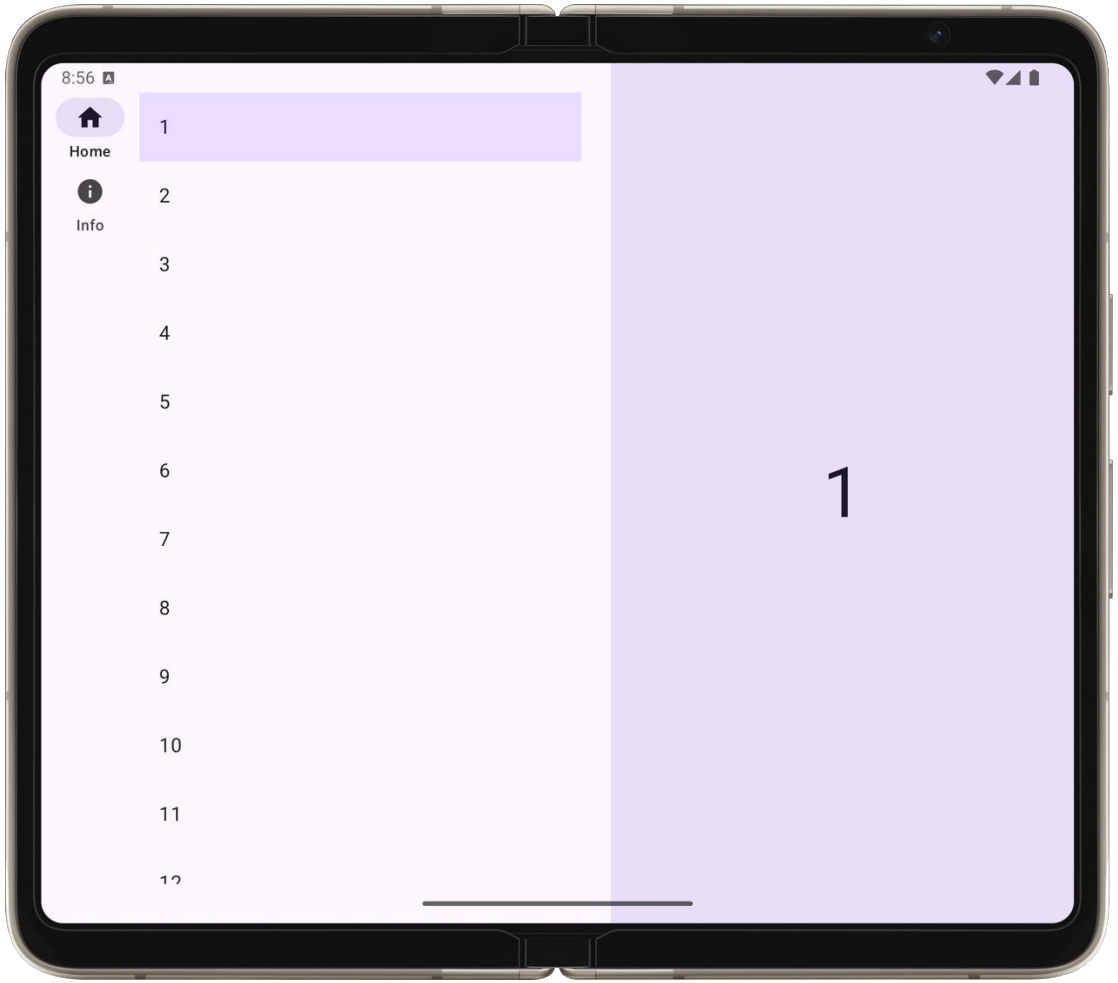

```
1  val density = LocalDensity.current
2  val customNavSuiteType =
3    if (with(density) { currentWindowSize().width.toDp() >= 1200.dp }) {
4      NavigationSuiteType.NavigationDrawer
5    } else {
6      NavigationSuiteScaffoldDefaults.calculateFromAdaptiveInfo(adaptiveInfo)
7    }
```

Decide based on the window size



Decide based on the window size class

```
1  val customNavSuiteType =
2    if (adaptiveInfo.windowSizeClass.windowWidthSizeClass == WindowWidthSizeClass.EXPANDED) {
3      NavigationSuiteType.NavigationDrawer
4    } else {
5      NavigationSuiteScaffoldDefaults.calculateFromAdaptiveInfo(adaptiveInfo)
6    }
```



Home pane: list and detail of current item

```kotlin
@OptIn(ExperimentalMaterial3AdaptiveApi::class)
@Composable
fun HomePane() {
    val navigator = rememberListDetailPaneScaffoldNavigator<Int>(
        scaffoldDirective = calculatePaneScaffoldDirectiveWithTwoPanesOnMediumWidth(
            currentWindowAdaptiveInfo()
        )
    )
    var currentIndex by rememberSaveable { mutableIntStateOf(-1) }
    val onItemClicked: (Int) -> Unit = { id ->
        currentIndex = id
        navigator.navigateTo(
            pane = ListDetailPaneScaffoldRole.Detail, content = id
        )
    }
    val detailVisible = navigator.scaffoldValue[ListDetailPaneScaffoldRole.Detail] == PaneAdaptedValue.Expanded
    if (detailVisible && currentIndex == -1) currentIndex = 0
    BackHandler(navigator.canNavigateBack()) { navigator.navigateBack() }
    ListDetailPaneScaffold(directive = navigator.scaffoldDirective,
        value = navigator.scaffoldValue,
        listPane = { MyList(onItemClicked, currentIndex, detailVisible) },
        detailPane = {
            MyListDetail(currentIndex = currentIndex,
                listHidden = navigator.scaffoldValue[ListDetailPaneScaffoldRole.List] == PaneAdaptedValue.Hidden,
                onBackClicked = { navigator.navigateBack() })
        })
}
```

```kotlin
@OptIn(ExperimentalMaterial3AdaptiveApi::class)
@Composable
fun HomePane() {
    val navigator = rememberListDetailPaneScaffoldNavigator<Int>(
        scaffoldDirective = calculatePaneScaffoldDirectiveWithTwoPanesOnMediumWidth(
            currentWindowAdaptiveInfo()
        )
    )
    var currentIndex by rememberSaveable { mutableIntStateOf(-1) }
    val onItemClicked: (Int) -> Unit = { id ->
        currentIndex = id
        navigator.navigateTo(
            pane = ListDetailPaneScaffoldRole.Detail, content = id
        )
    }
    val detailVisible = navigator.scaffoldValue[ListDetailPaneScaffoldRole.Detail] == PaneAdaptedValue.Expanded
    if (detailVisible && currentIndex == -1) currentIndex = 0
    BackHandler(navigator.canNavigateBack()) { navigator.navigateBack() }
    ListDetailPaneScaffold(directive = navigator.scaffoldDirective,
        value = navigator.scaffoldValue,
        listPane = { MyList(onItemClicked, currentIndex, detailVisible) },
        detailPane = {
            MyListDetail(currentIndex = currentIndex,
                listHidden = navigator.scaffoldValue[ListDetailPaneScaffoldRole.List] == PaneAdaptedValue.Hidden,
                onBackClicked = { navigator.navigateBack() })
        }
    )
}
```

How the scaffold should arrange its panes

Indicates how each pane of the scaffold is adapted

```
@ExperimentalMaterial3AdaptiveApi
@Composable
fun ListDetailPaneScaffold(
    directive: PaneScaffoldDirective,
    value: ThreePaneScaffoldValue,
    listPane: @Composable ThreePaneScaffoldScope.() -> Unit,
    detailPane: @Composable ThreePaneScaffoldScope.() -> Unit,
    modifier: Modifier = Modifier,
    extraPane: (@Composable ThreePaneScaffoldScope.() -> Unit)? = null,
) {
    ThreePaneScaffold(
        modifier = modifier.fillMaxS
        scaffoldDirective = directiv
        scaffoldValue = value,
        paneOrder = ListDetailPaneSc
        secondaryPane = listPane,
        tertiaryPane = extraPane,
        primaryPane = detailPane
    )
}
```

`primary` - `PaneAdaptedValue` of the primary pane of `ThreePaneScaffold`
`secondary` - `PaneAdaptedValue` of the secondary pane of `ThreePaneScaffold`
`tertiary` - `PaneAdaptedValue` of the tertiary pane of `ThreePaneScaffold`

Contains the adapted values of each pane

```
Denotes that the associated pane should be displayed in its full width and height.
val Expanded = PaneAdaptedValue( description: "Expanded")
Denotes that the associated pane should be hidden.
val Hidden = PaneAdaptedValue( description: "Hidden")
```

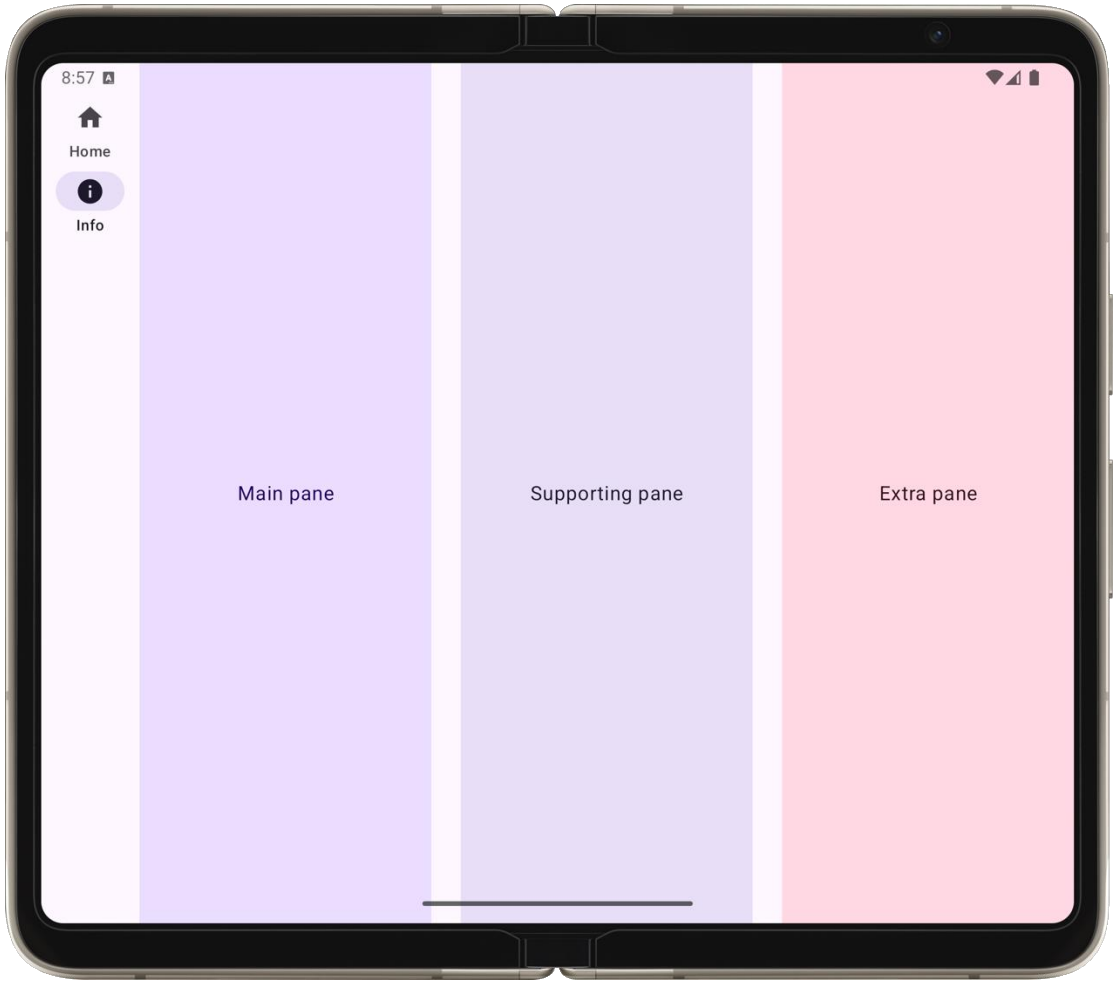
```kotlin
@OptIn(ExperimentalMaterial3AdaptiveApi::class)
@Composable
fun ThreePaneScaffoldScope.MyList(
    onItemClicked: (Int) -> Unit, currentIndex: Int, detailVisible: Boolean
) {
    AnimatedPane {
        LazyColumn(modifier = Modifier.safeDrawingPadding()) {
            items(20) {
                ListItem(
                    headlineContent = { Text("${it + 1}") },
                    modifier = Modifier.clickable { onItemClicked(it) },
                    colors = when (detailVisible) {
                        false -> ListItemDefaults.colors()
                        true -> if (currentIndex == it) {
                            ListItemDefaults.colors(
                                containerColor = MaterialTheme.colorScheme.primaryContainer,
                                headlineColor = MaterialTheme.colorScheme.onPrimaryContainer
                            )
                        } else { ListItemDefaults.colors()}
                    }
                )
}}}}
```

```
navigator.scaffoldValue[ListDetailPaneScaffoldRole.Detail] == PaneAdaptedValue.Expanded
```

```
object ListDetailPaneScaffoldRole {

    The list pane of ListDetailPaneScaffold, which is supposed to hold a list of item summaries
    that can be selected from, for example, the inbox mail list of a mail app. It maps to
    ThreePaneScaffoldRole.Secondary.

    val List = ThreePaneScaffoldRole.Secondary


    The detail pane of ListDetailPaneScaffold, which is supposed to hold the detailed info of a
    selected item, for example, the mail content currently being viewed. It maps to
    ThreePaneScaffoldRole.Primary.

    val Detail = ThreePaneScaffoldRole.Primary


    The extra pane of ListDetailPaneScaffold, which is supposed to hold any supplementary info
    besides the list and the detail panes, for example, a task list or a mini-calendar view of a mail app.
    It maps to ThreePaneScaffoldRole.Tertiary.

    val Extra = ThreePaneScaffoldRole.Tertiary
}
```

```
1  @OptIn(ExperimentalMaterial3AdaptiveApi::class)
2  @Composable
3  fun ThreePaneScaffoldScope.MyListDetail(
4      currentIndex: Int, listHidden: Boolean, onBackClicked: () -> Unit
5  ) {
6      AnimatedPane {
7          Box(
8              contentAlignment = Alignment.Center,
9              modifier = Modifier.background(MaterialTheme.colorScheme.secondaryContainer)
10         ) {
11             Text(
12                 "${currentIndex + 1}",
13                 style = MaterialTheme.typography.displayLarge,
14                 color = MaterialTheme.colorScheme.onSecondaryContainer
15             )
16             if (listHidden) {
17                 IconButton(
18                     onClick = onBackClicked,
19                     modifier = Modifier
20                         .align(Alignment.TopStart)
21                         .safeContentPadding()
22                 ) {
23                     Icon(Icons.AutoMirrored.Outlined.ArrowBack,
24                         contentDescription = stringResource(R.string.back))
25 }}}}}
```

```
navigator.scaffoldValue[ListDetailPaneScaffoldRole.List] == PaneAdaptedValue.Hidden,
```



Example of a supporting pane

```kotlin
@OptIn(ExperimentalMaterial3AdaptiveApi::class)
@Composable
fun InfoPane() {
    val navigator = rememberSupportingPaneScaffoldNavigator(
        scaffoldDirective = calculatePaneScaffoldDirective(currentWindowAdaptiveInfo()).copy(
            maxHorizontalPartitions = when (currentWindowAdaptiveInfo().windowSizeClass.windowWidthSizeClass) {
                WindowWidthSizeClass.MEDIUM -> 2
                WindowWidthSizeClass.EXPANDED -> 3
                else -> 1
            }
        )
    )
    BackHandler(navigator.canNavigateBack()) {
        navigator.navigateBack()
    }
    SupportingPaneScaffold(
        directive = navigator.scaffoldDirective,
        mainPane = { MainPane(navigator = navigator) },
        supportingPane = { SupportingPane(navigator = navigator) },
        extraPane = { ExtraPane() },
        value = navigator.scaffoldValue
    )
}
```

```kotlin
@ExperimentalMaterial3AdaptiveApi
@Composable
fun SupportingPaneScaffold(
    directive: PaneScaffoldDirective,
    value: ThreePaneScaffoldValue,
    mainPane: @Composable ThreePaneScaffoldScope.() -> Unit,
    supportingPane: @Composable ThreePaneScaffoldScope.() -> Unit,
    modifier: Modifier = Modifier,
    extraPane: (@Composable ThreePaneScaffoldScope.() -> Unit)? = null,
) {
    ThreePaneScaffold(
        modifier = modifier.fillMaxSize(),
        scaffoldDirective = directive,
        scaffoldValue = value,
        paneOrder = SupportingPaneScaffoldDefaults.PaneOrder,
        secondaryPane = supportingPane,
        tertiaryPane = extraPane,
        primaryPane = mainPane
    )
}
```

```kotlin
@OptIn(ExperimentalMaterial3AdaptiveApi::class)
@Composable
fun ThreePaneScaffoldScope.MainPane(navigator: ThreePaneScaffoldNavigator<Nothing>) {
    ColoredBox(textColor = MaterialTheme.colorScheme.onPrimaryContainer,
        backgroundColor = MaterialTheme.colorScheme.primaryContainer,
        resIdMessage = R.string.main_pane,
        resIdButton = R.string.show_supporting_pane,
        shouldShowButton = navigator.scaffoldValue[SupportingPaneScaffoldRole.Supporting] == PaneAdaptedValue.Hidden,
        onClick = { navigator.navigateTo(SupportingPaneScaffoldRole.Supporting) })
}
```

```kotlin
@OptIn(ExperimentalMaterial3AdaptiveApi::class)
@Composable
private fun ThreePaneScaffoldScope.ColoredBox(textColor: Color,
    backgroundColor: Color, shouldShowButton: Boolean,
    @StringRes resIdMessage: Int, @StringRes resIdButton: Int,
    onClick: () -> Unit = {}
) {
    AnimatedPane {
        Box(
            modifier = Modifier.background(backgroundColor), contentAlignment = Alignment.BottomEnd
        ) {
            Text(
                text = stringResource(resIdMessage),
                style = MaterialTheme.typography.bodyLarge,
                textAlign = TextAlign.Center,
                modifier = Modifier.align(Alignment.Center),
                color = textColor
            )
            if (shouldShowButton) {
                Button(onClick = onClick, modifier = Modifier.padding(all = 32.dp)) { Text(text = stringResource(resIdButton)) }
            }
        }
    }
}
```

```kotlin
@OptIn(ExperimentalMaterial3AdaptiveApi::class)
@Composable
fun ThreePaneScaffoldScope.SupportingPane(navigator: ThreePaneScaffoldNavigator<Nothing>) {
    ColoredBox(textColor = MaterialTheme.colorScheme.onSecondaryContainer,
        backgroundColor = MaterialTheme.colorScheme.secondaryContainer,
        resIdMessage = R.string.supporting_pane,
        resIdButton = R.string.show_main_pane,
        shouldShowButton = navigator.scaffoldValue[SupportingPaneScaffoldRole.Main] == PaneAdaptedValue.Hidden,
        onClick = { navigator.navigateTo(SupportingPaneScaffoldRole.Main) })
}
```

```kotlin
@OptIn(ExperimentalMaterial3AdaptiveApi::class)
@Composable
fun ThreePaneScaffoldScope.ExtraPane() {
    ColoredBox(
        textColor = MaterialTheme.colorScheme.onTertiaryContainer,
        backgroundColor = MaterialTheme.colorScheme.tertiaryContainer,
        resIdMessage = R.string.extra_pane,
        resIdButton = -1,
        shouldShowButton = false
    )
}
```

Conclusion

- Material3 Adaptive greatly simplifies layout on large screens
- … and smartphones …
- The API feels intuitive to use
- Can be customized easily
- There’s no excuse for not using it, at least in new apps
- Mileage may vary for existing apps

https://github.com/tkuenneth/foldables_and_large_screens

Thank you

 @tkuenneth

 @tkuenneth

 @tkuenneth@snapp.social

Experts

SnappMobile