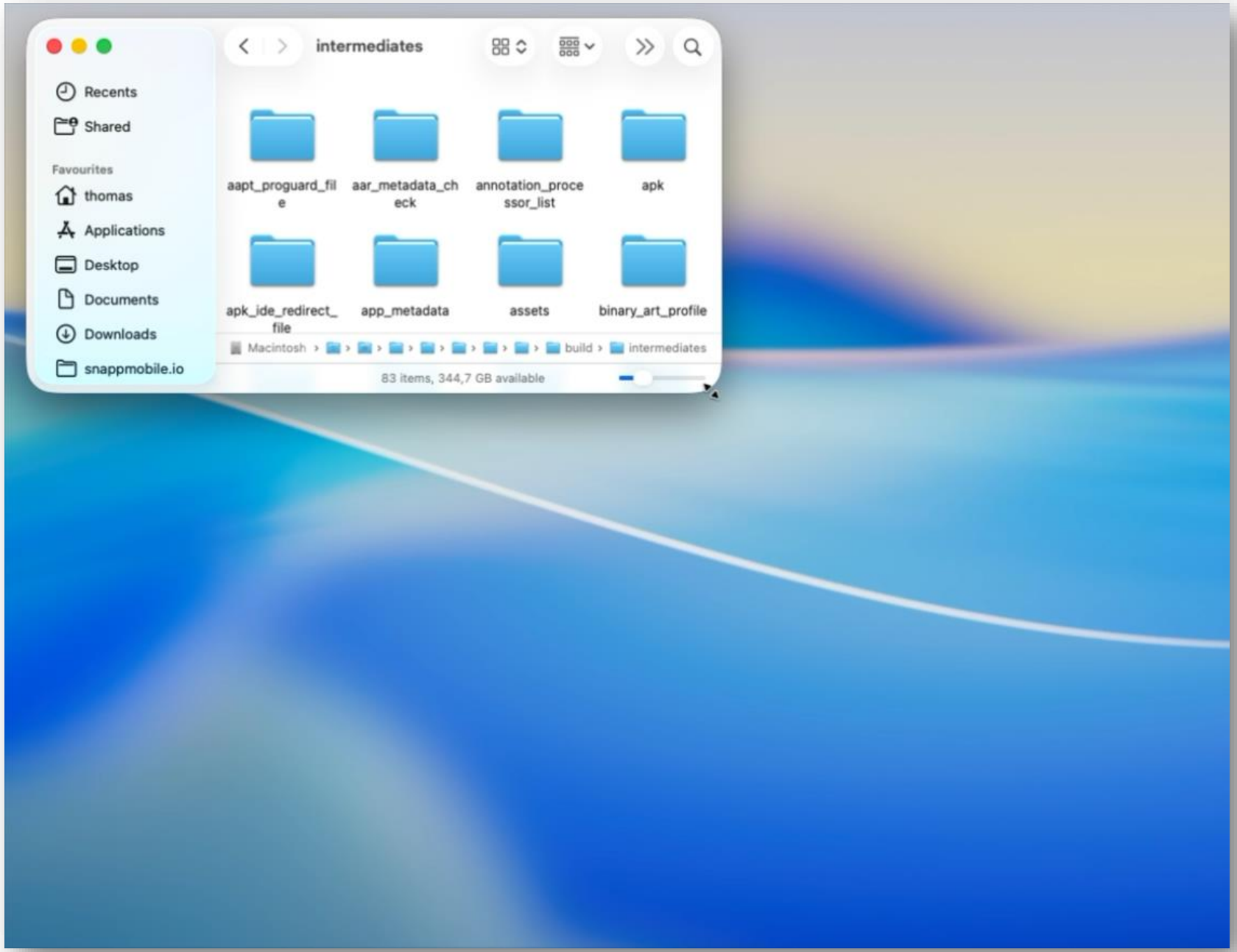
Building Responsive User Interfaces with Compose Multiplatform

Thomas Künneth

Experts

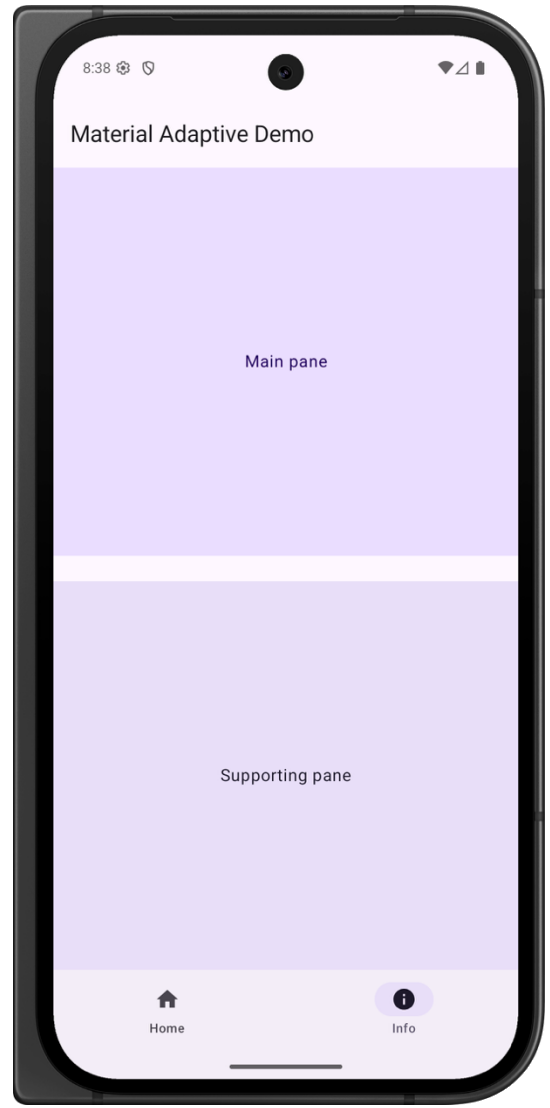


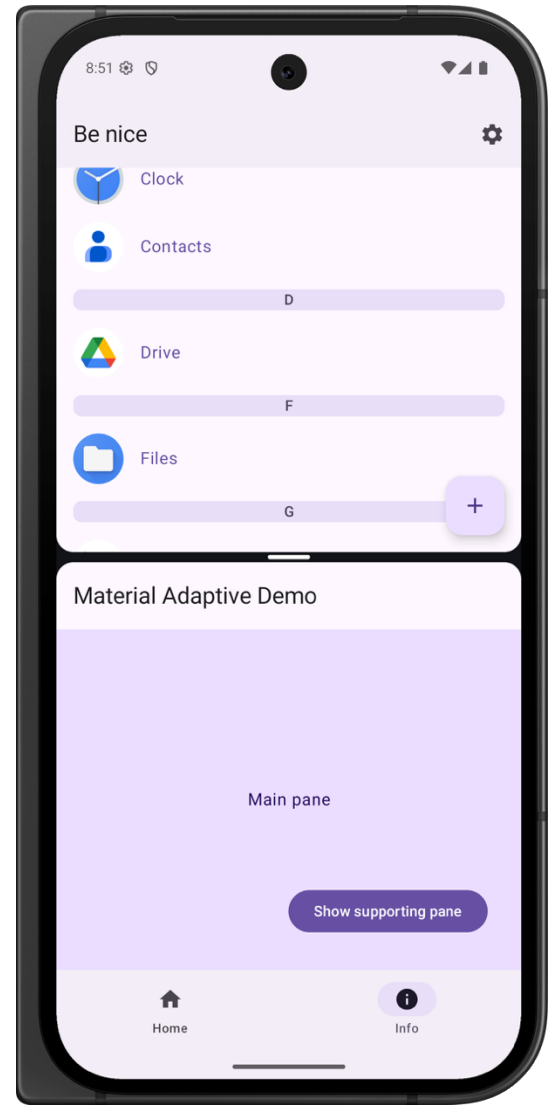
Experts

SnappMobile

Experts

SnappMobile



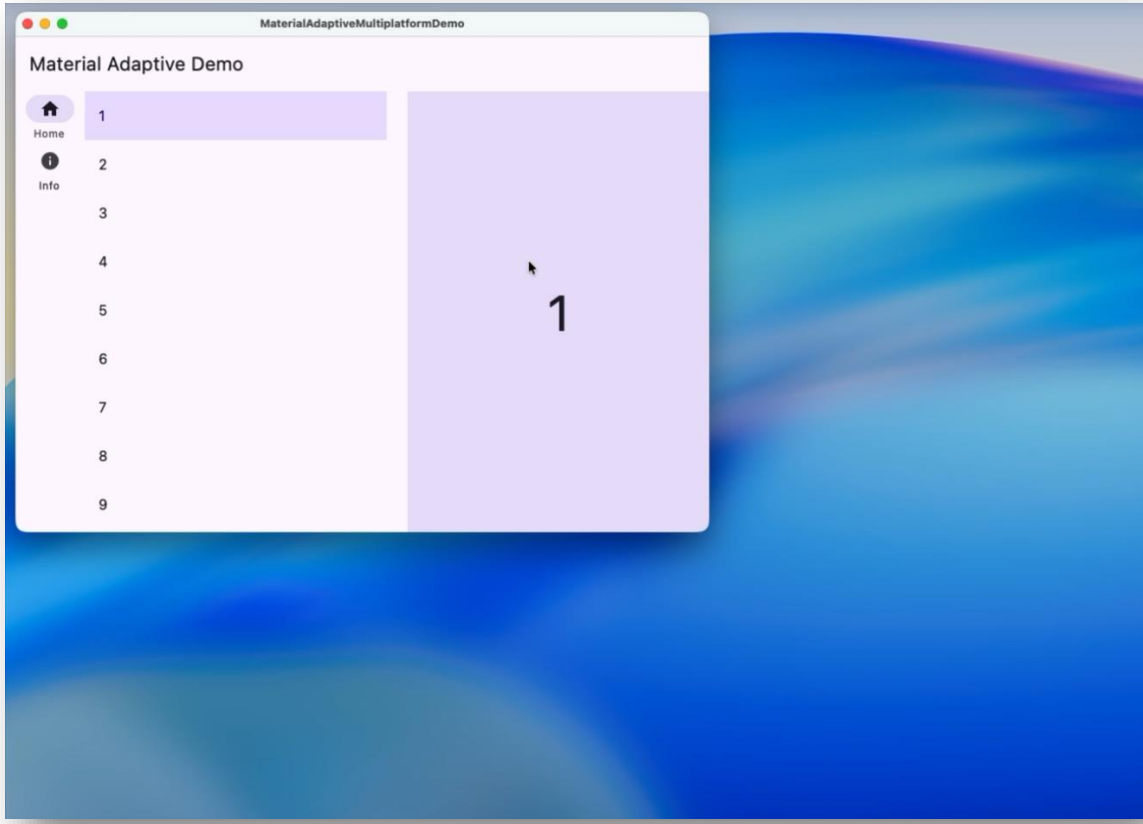


Experts

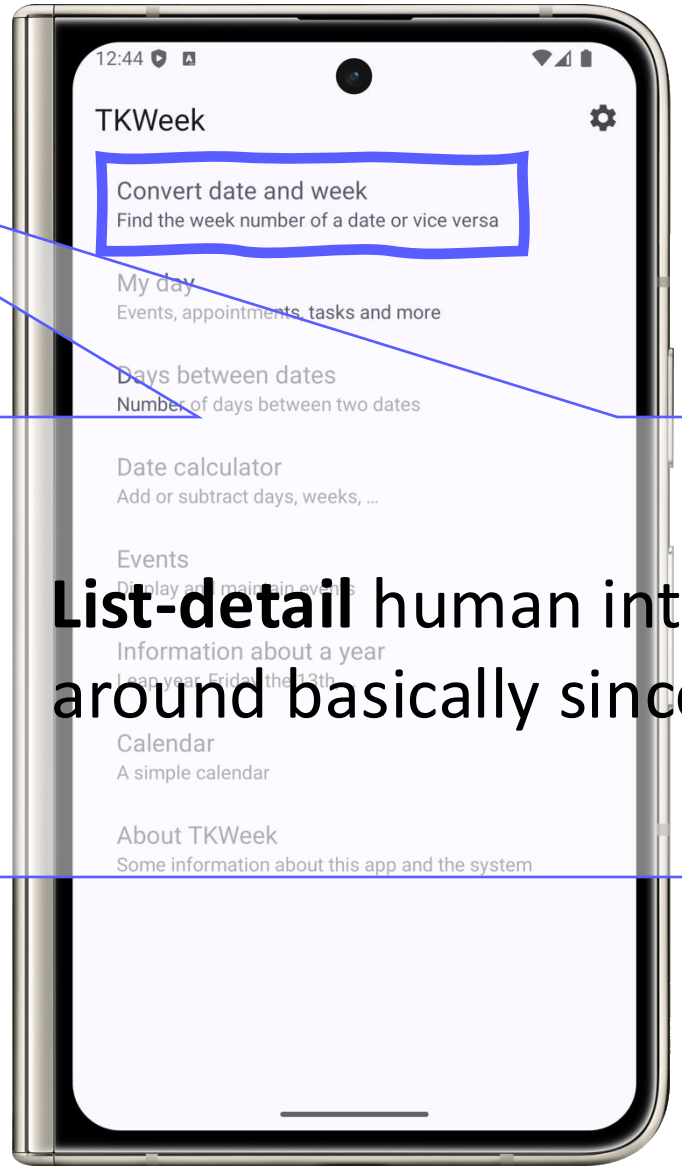
SnappMobile

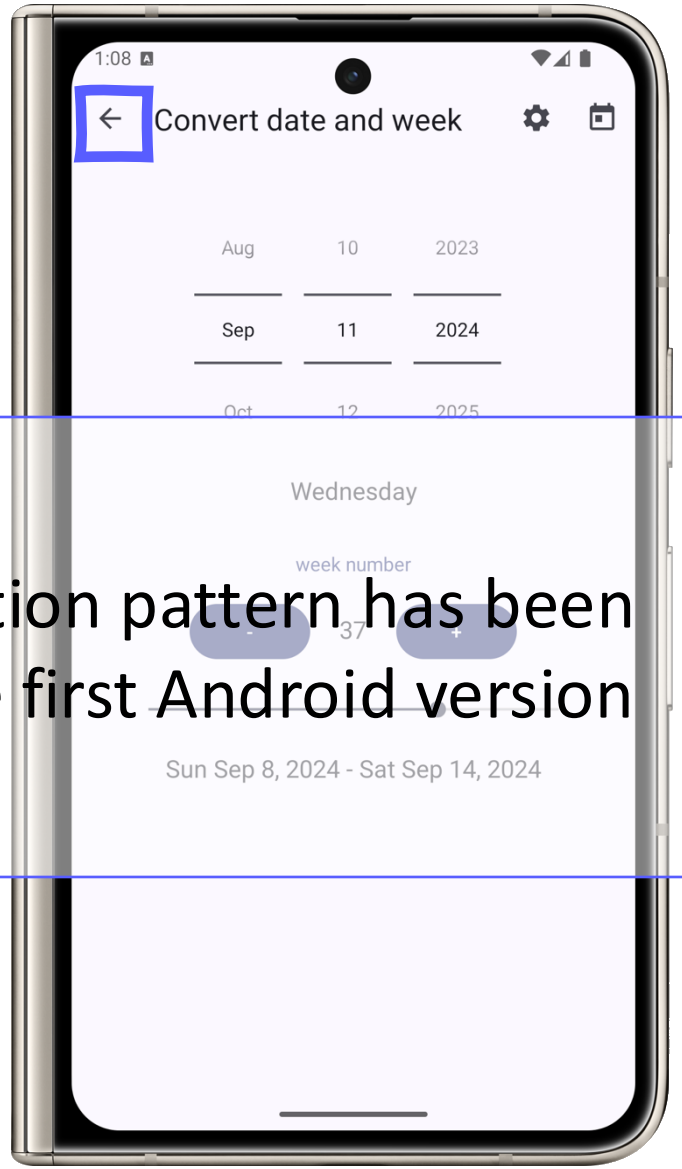
Compose Multiplatform

- Cross-platform framework for building native apps for Android, iOS, the Desktop, and the Web
- Uses Kotlin for business logic and UI
- UI is built using Jetpack Compose



Since Compose Multiplatform uses Jetpack Compose, let’s take a look how modern Android apps tackle different **screen sizes** and **form factors**

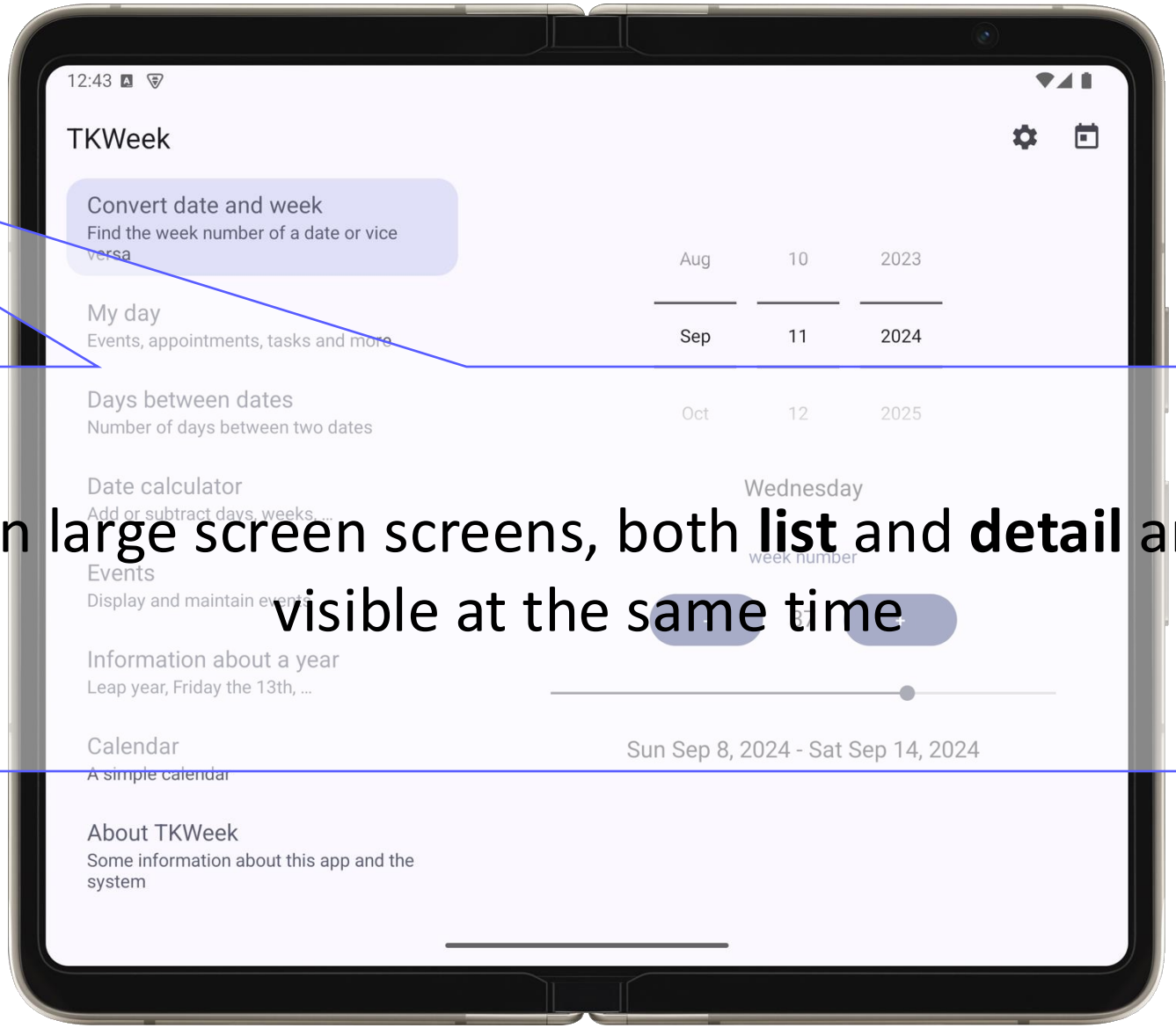




List-detail human interaction pattern has been around basically since the first Android version

Experts

SnappMobile



On large screen screens, both **list** and **detail** are visible at the same time

Experts

SnappMobile

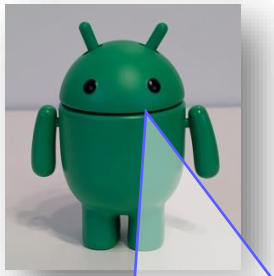
Granted, that’s pretty much how we do it basically everywhere

Just took Google some time to formalize and provide architectural blueprints and library support

Material 3

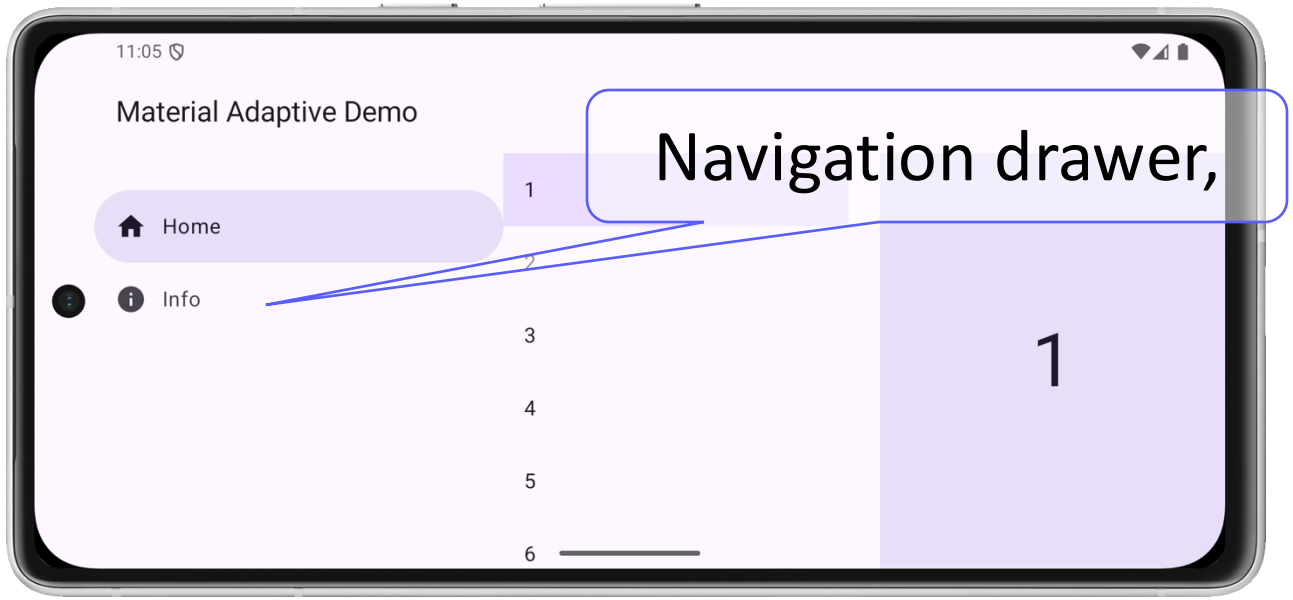
- Google’s design system and design language on Android
- Used in many Google Web Apps and Flutter
- Also available in Compose Multiplatform

- Defines components and their behaviour
- Specifies visual appearance like fonts, colours, paddings, margins and gaps
- Defines layout, including composition of element hierarchies
- Provides navigation

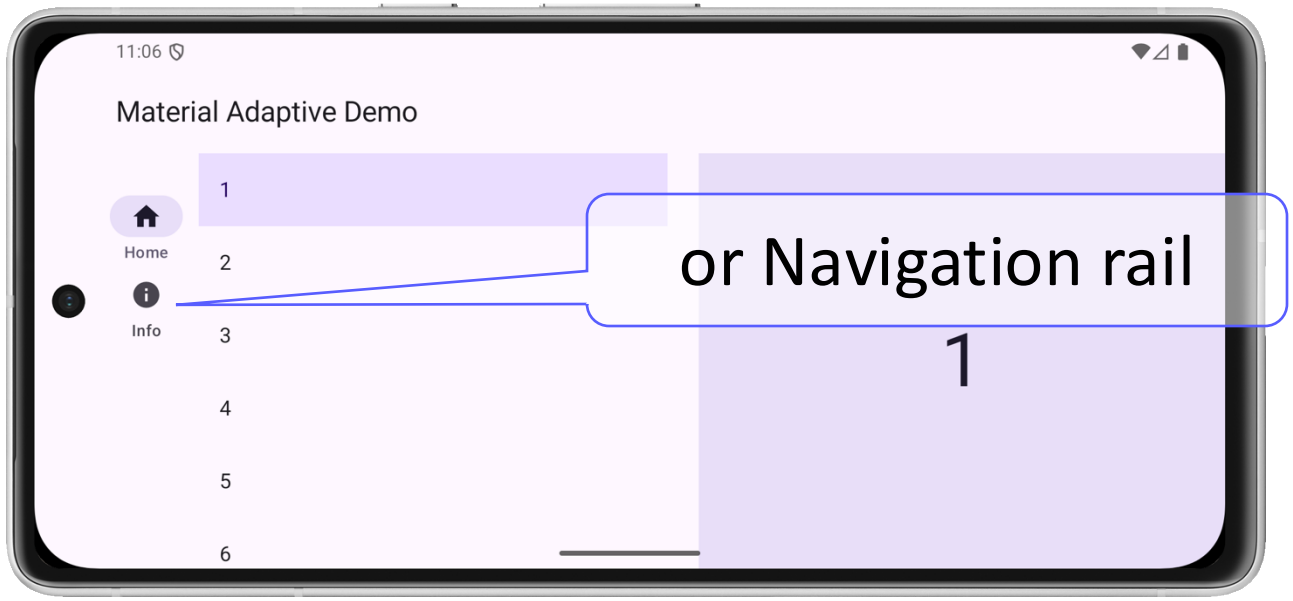


Top-level navigation is done using either …

Navigation bar,



Navigation drawer,



or Navigation rail

Experts

SnappMobile

But when to use which

?

Material3 Adaptive

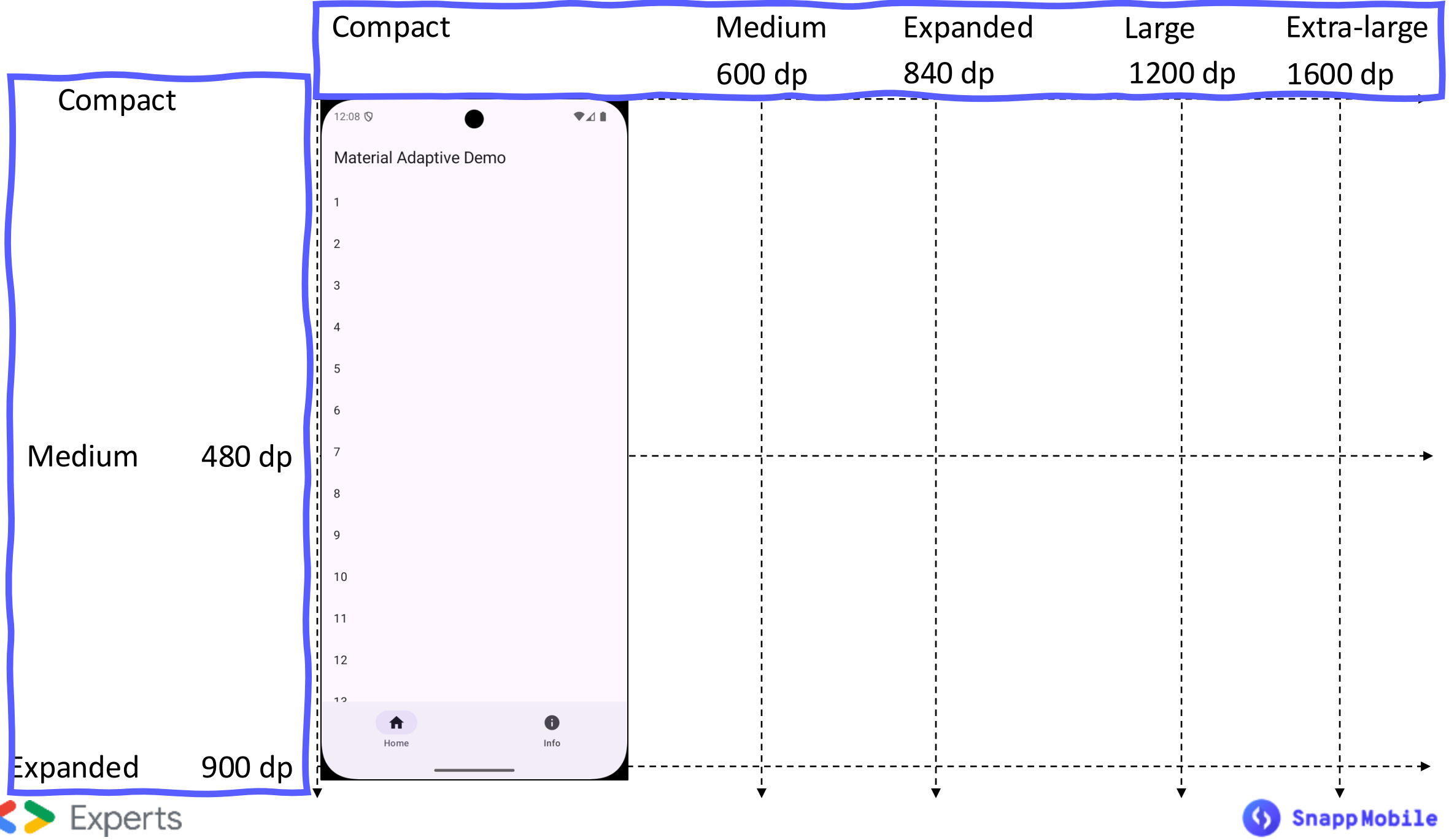
- Builds upon **canonical layouts** to create consistent, responsive user interfaces
- **Selects** the best **navigation component** for the available space
- **Provides pre-built scaffolds** to simplify implementation

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

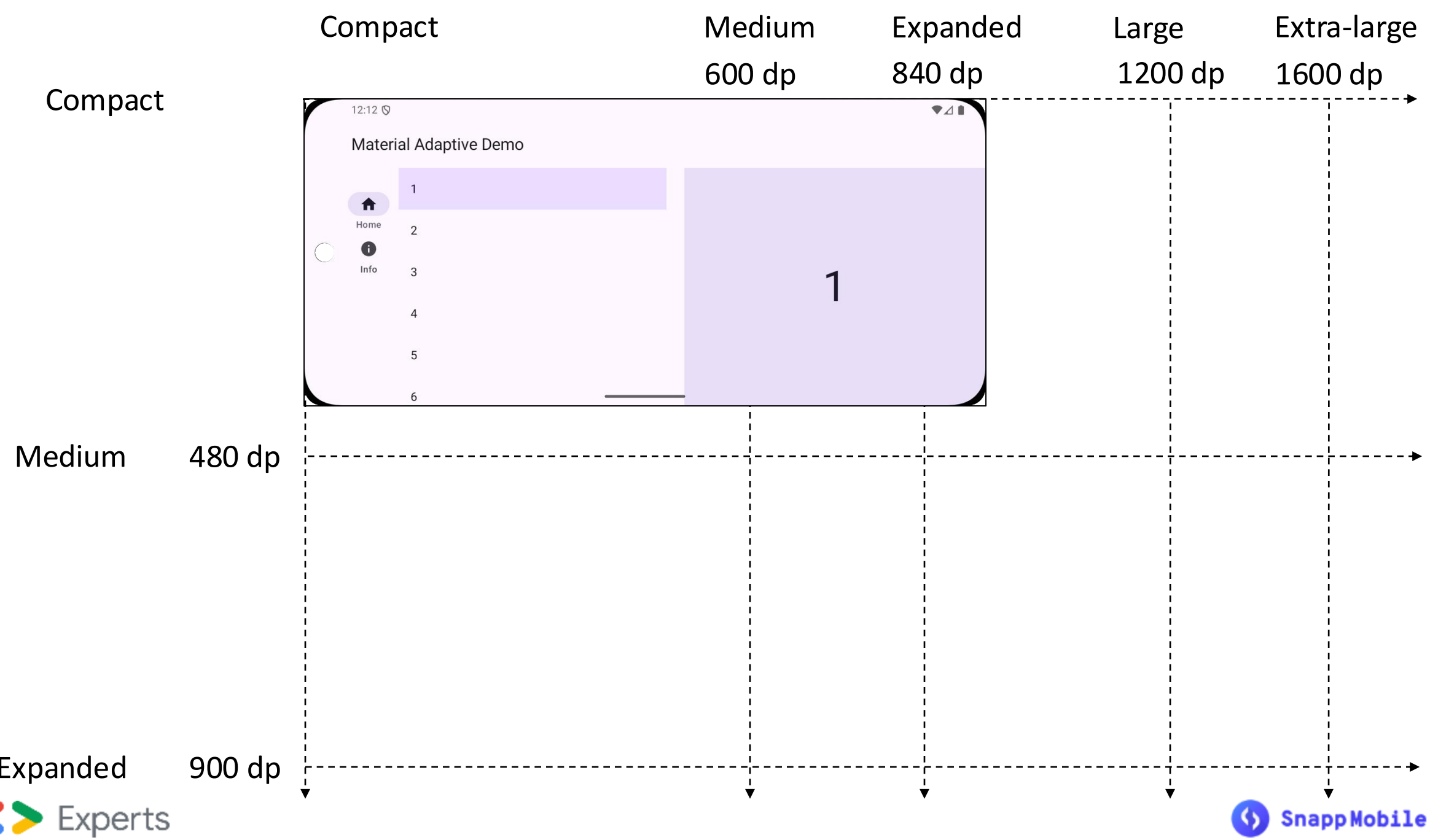
Window size classes

- A set of opinionated layout breakpoints
- Classify available width and height separately
- Define categories: **Compact**, **Medium**, **Expanded**, **Large**, and **Extra-large**
- Affect the location and size of both **navigation and content area**, as well as which components are used



Experts

SnappMobile



Experts

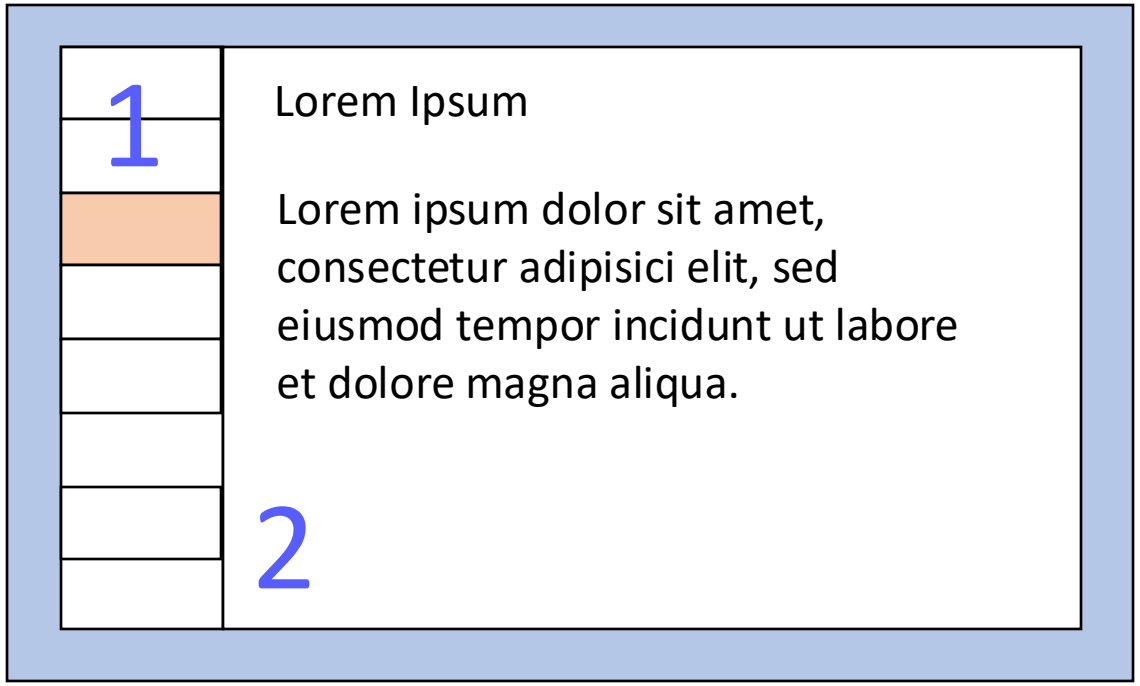
SnappMobile

Canonical layouts

SnappMobile

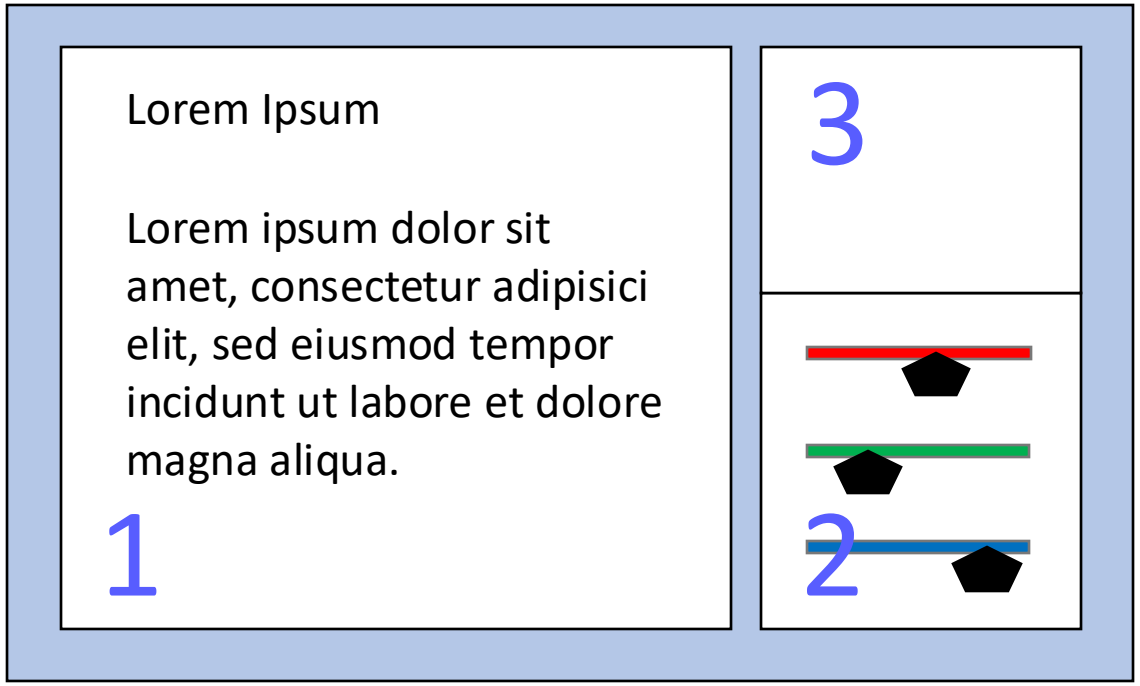
List-detail

- Scrollable lists of items
- Item detail containing supplementary information
- Two logical panes



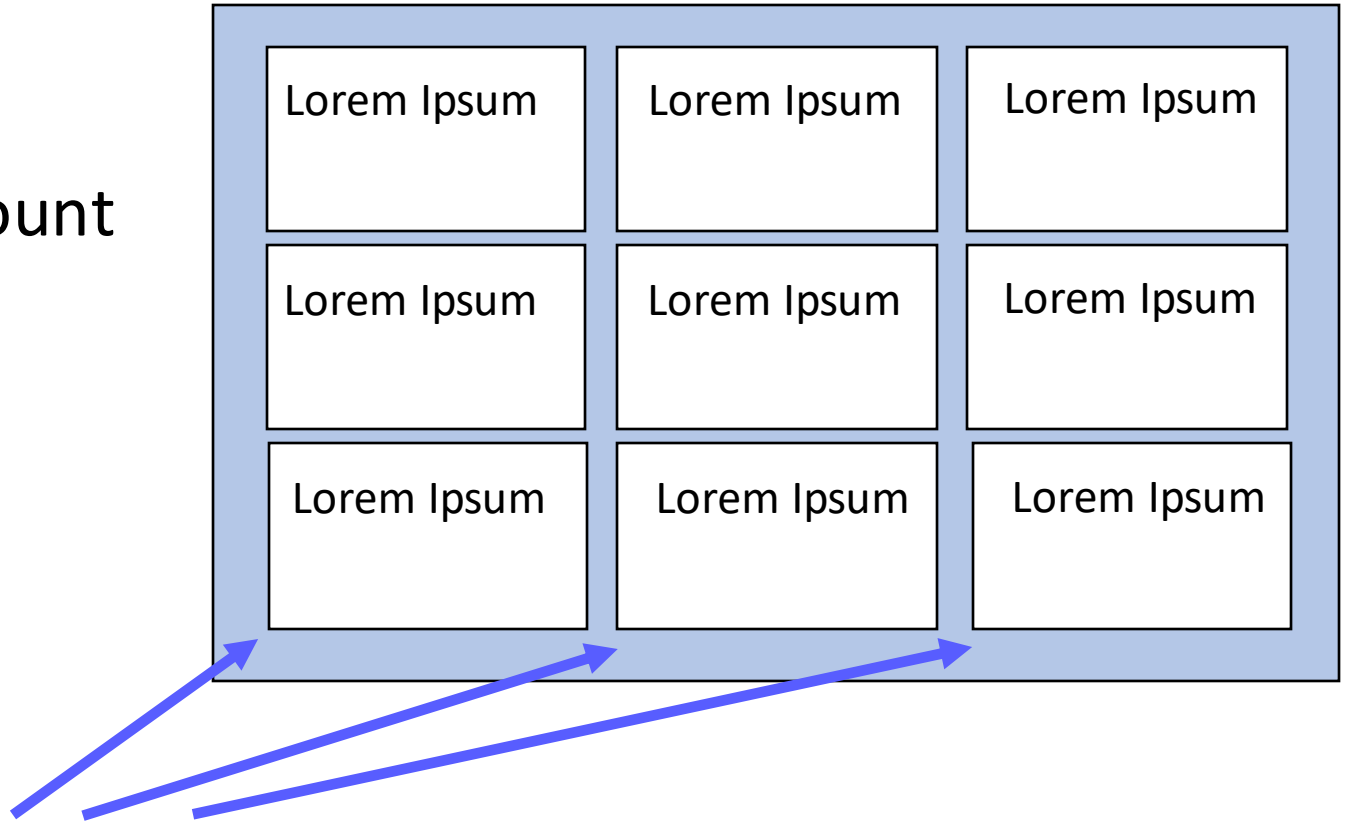
Supporting pane

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



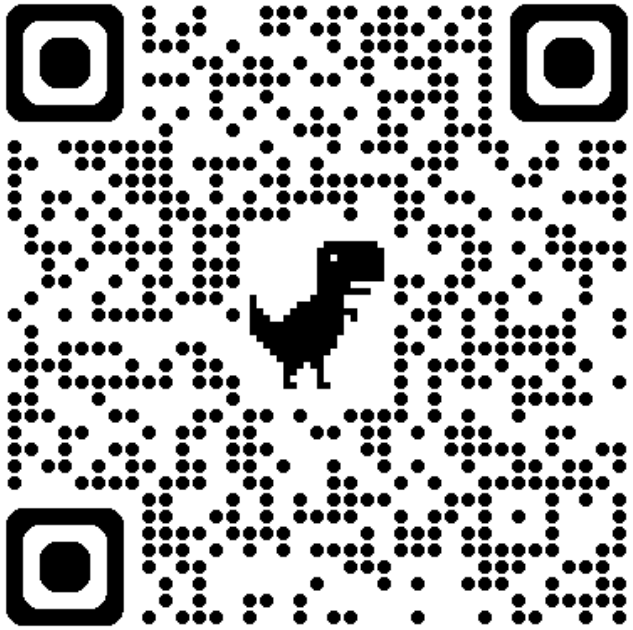
Feed

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



These are no panes

- Material 3 Adaptive shows or hides panes based on Window Size classes
- Provides means to navigate between panes
- *List-detail* implemented by `ListDetailPaneScaffold` and `NavigableListDetailPaneScaffold`
- *Supporting Pane* implemented by `SupportingPaneScaffold` and `NavigableSupportingPaneScaffold`
- No implementation for *Feed*



https://github.com/tkuenneth/foldables_and_large_screens

Experts

```
package de.thomaskuenneth.material.adaptive.multiplatform

import ...

@OptIn( ...markerClass = ExperimentalMaterial3Api::class) 3 Usages
@Composable
fun MaterialAdaptiveDemo() {
    var currentDestination by rememberSaveable { mutableStateOf( val
    val adaptiveInfo = currentWindowAdaptiveInfo()
    val customNavSuiteType =
        if (adaptiveInfo.windowSizeClass.isWidthAtLeastBreakpoint( v
            NavigationSuiteType.NavigationDrawer
        } else {
            NavigationSuiteScaffoldDefaults.calculateFromAdaptiveIn
        }
    val navigationState = remember { NavigationState() }
    Scaffold(
        contentWindowInsets = WindowInsets(),
        topBar = {
            TopAppBar(
                title = { Text( text = stringResource( resource = Res.s
                navigationIcon = {
                    if (navigationState.canNavigateBack) {
                        IconButton(
                            onClick = {
                                navigationState.navigateBack()
                            }
                        ) {
                            Icon(
                                imageVector = Icons.AutoMirrored.Out
```

```kotlin
fun main() = application {
    Window(
        onCloseRequest = ::exitApplication,
        title = stringResource(Res.string.app_name),
    ) {
        MaterialAdaptiveDemo()
    }
}
```

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

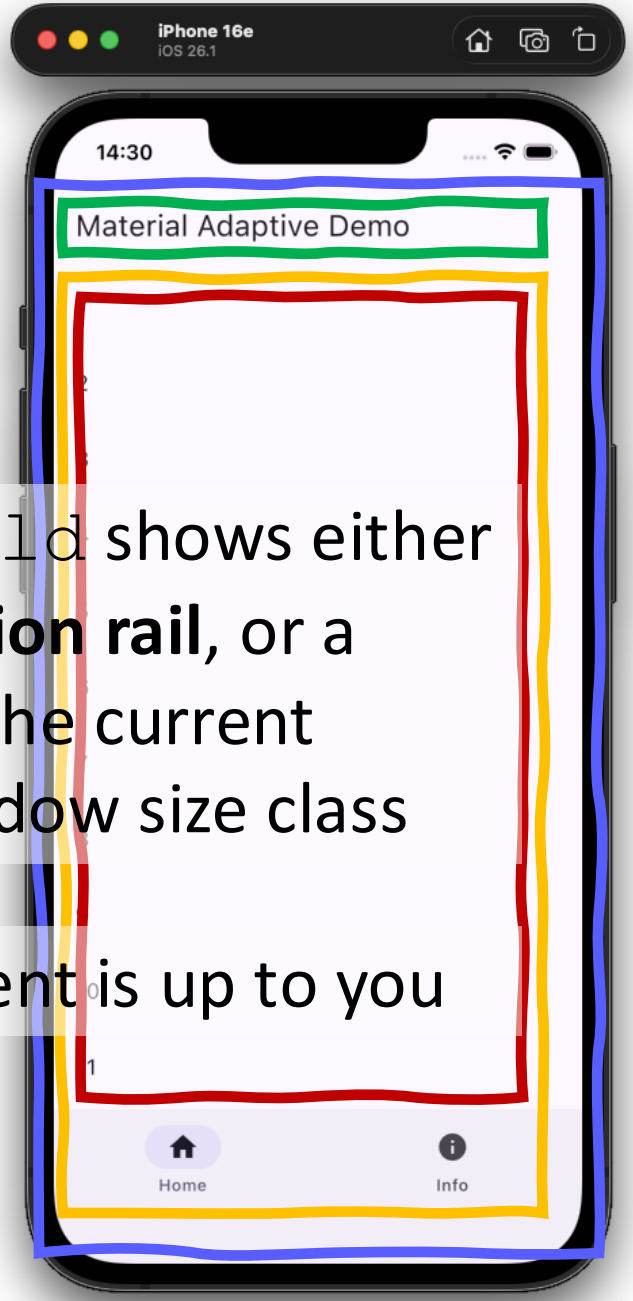
Sets a color scheme and invokes `MaterialAdaptiveDemoScaffold`

```kotlin
package de.thomaskuenneth.material.adaptive.multiplatform

import androidx.compose.ui.window.ComposeUIViewController

fun MainViewController() = ComposeUIViewController { MaterialAdaptiveDemo() }
```

Experts

`NavigationSuiteScaffold` shows either a **navigation drawer**, a **navigation rail**, or a **navigation bar**, depending on the current horizontal and / or vertical window size class

What will be shown as its content is up to you




```kotlin
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MaterialAdaptiveDemoScaffold() {
    var currentDestination by rememberSaveable { mutableStateOf(AppDestinations.Home) }
    val adaptiveInfo = currentWindowAdaptiveInfo()
    val customNavSuiteType =
        if (adaptiveInfo.windowSizeClass.isWidthAtLeastBreakpoint(WIDTH_DP_EXPANDED_LOWER_BOUND)) {
            NavigationSuiteType.NavigationDrawer
        } else {
            NavigationSuiteScaffoldDefaults.calculateFromAdaptiveInfo(adaptiveInfo)
        }
    val navigationState = remember { NavigationState() }
    Scaffold(
        contentWindowInsets = WindowInsets(),
        topBar = {
            TopAppBar(
                title = { Text(stringResource(Res.string.app_name)) },
                navigationIcon = {
                    if (navigationState.canNavigateBack) {
                        IconButton(
                            onClick = {
                                navigationState.navigateBack()
                            }
                        ) {
                            Icon(
                                Icons.AutoMirrored.Outlined.ArrowBack,
                                contentDescription = stringResource(Res.string.back)
                            )
                        }
                    }
                }
            )
        }) { paddingValues ->
        NavigationSuiteScaffold(
            modifier = Modifier.padding(paddingValues),
            layoutType = customNavSuiteType,
            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(navigationState)
                }

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

Experts

are passed to `NavigationSuiteScaffold`

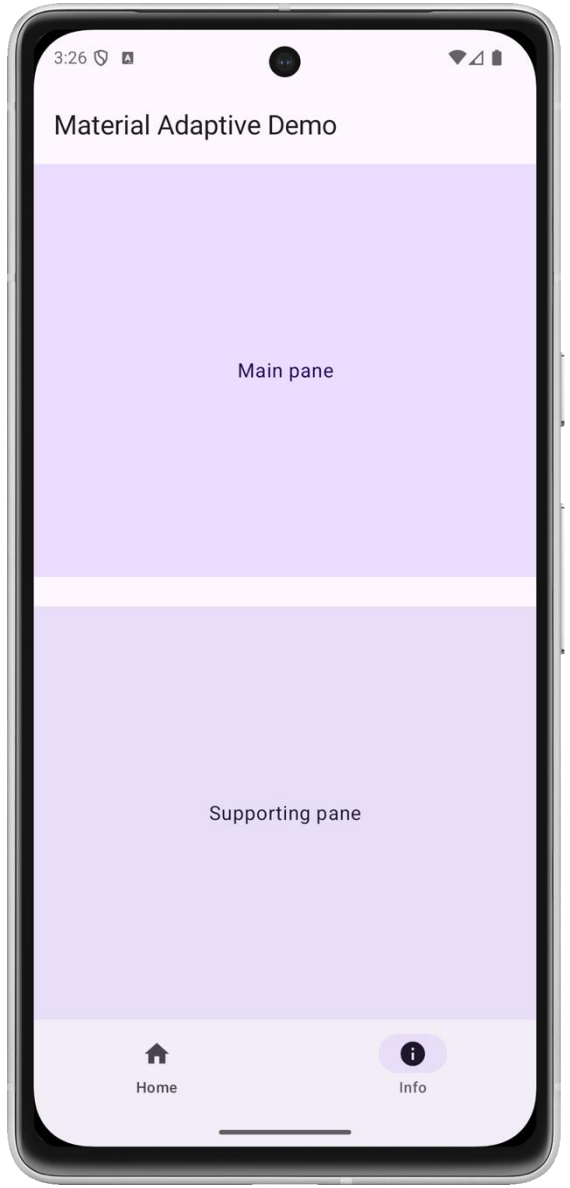
```kotlin
enum class AppDestinations(
    val labelRes: StringResource,
    val icon: ImageVector,
    val contentDescription: StringResource = labelRes,
) {
    Home(
        Res.string.tab_home,
        Icons.Default.Home
    ),
    Info(
        Res.string.tab_info,
        Icons.Default.Info
   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

Two top-level destinations

```kotlin
NavigationSuiteScaffold(
    modifier = Modifier.padding(paddingValues),
    layoutType = customNavSuiteType,
    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(navigationState)
        }

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

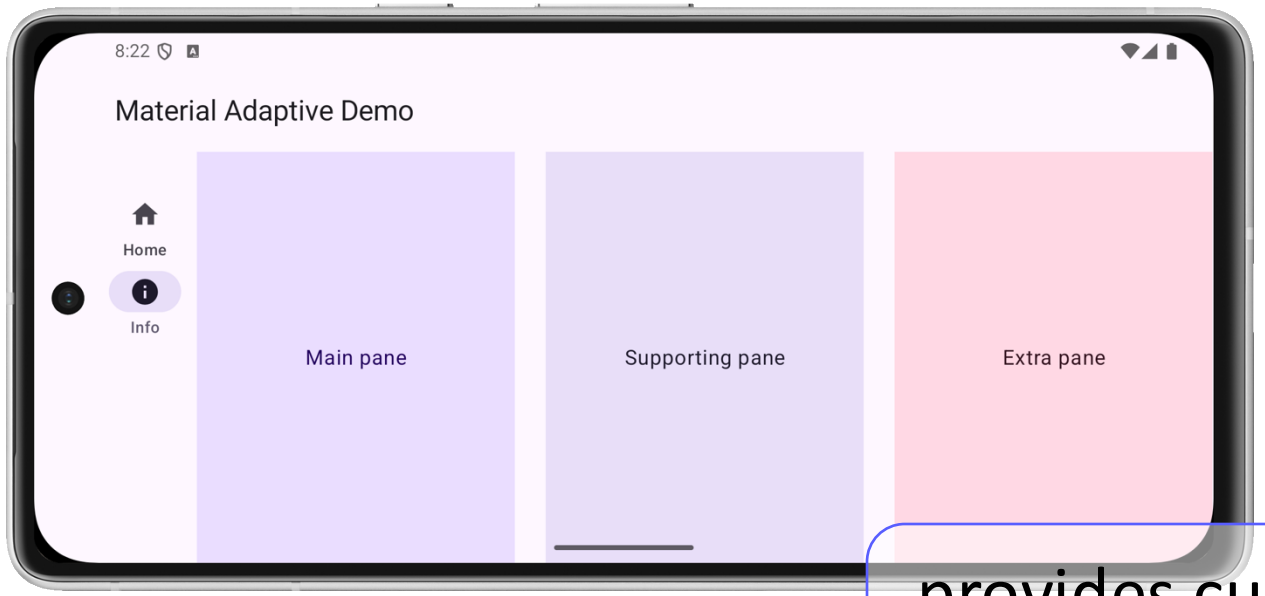
```kotlin
var currentDestination by rememberSaveable { mutableStateOf( value = AppDestinations.Home) }
```

Experts



Experts

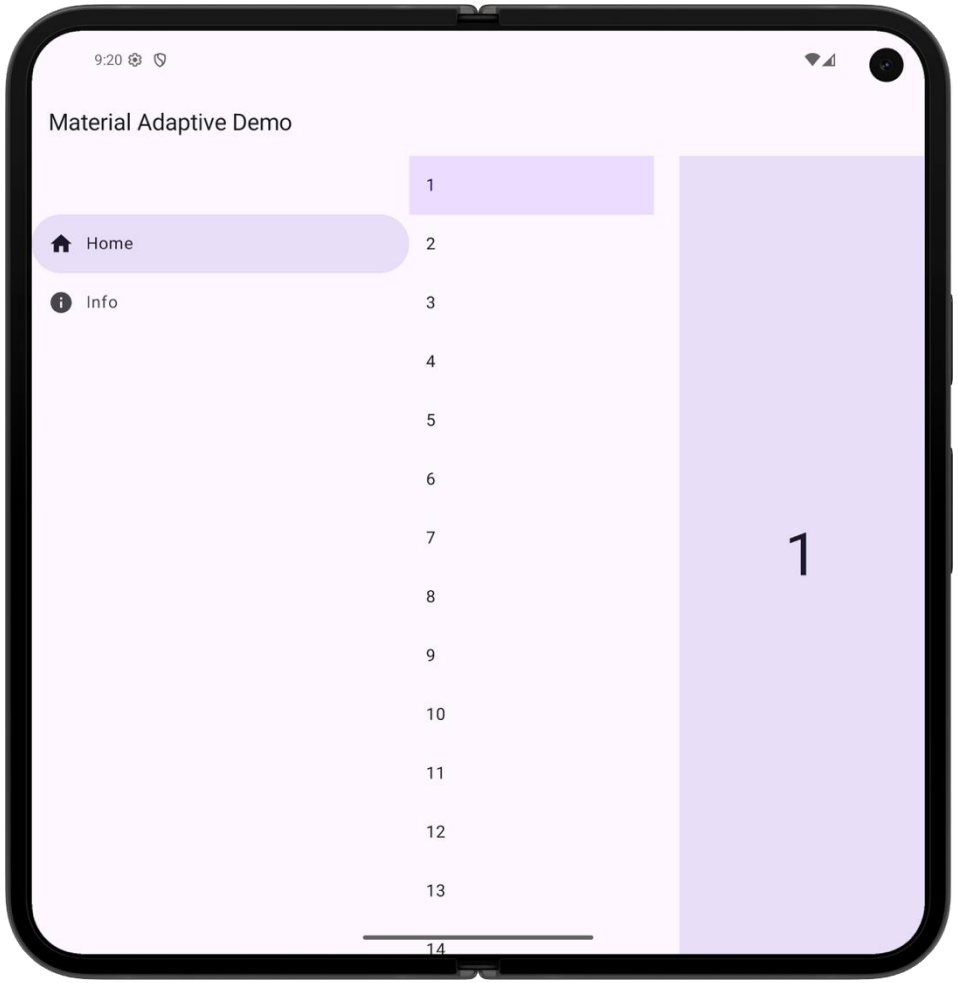
SnappMobile




```
1  val adaptiveInfo = currentWindowAdaptiveInfo()
2  val customNavSuiteType =
3      if (adaptiveInfo.windowSizeClass.isWidthAtLeastBreakpoint(WIDTH_DP_EXPANDED_LOWER_BOUND)) {
4          NavigationSuiteType.NavigationRail
5      } else {
6          NavigationSuiteScaffoldDefaults.calculateFromAdaptiveInfo(adaptiveInfo)
7      }
8  Scaffold(…) { paddingValues ->
9      NavigationSuiteScaffold(
10         modifier = Modifier.padding(paddingValues),
11         layoutType = customNavSuiteType,
12         navigationSuiteItems = {
13             …
```

provides current Window Size Class and device posture

Use a custom one or go with the default?



Home pane: list and detail of current item

```kotlin
@Composable
fun HomePane(navigationState: NavigationState) {
    val coroutineScope = rememberCoroutineScope()
    val navigator = rememberListDetailPaneScaffoldNavigator<Int>(
        scaffoldDirective = calculatePaneScaffoldDirectiveWithTwoPanesOnMediumWidth(
            currentWindowAdaptiveInfo()
        ))
    var currentIndex by rememberSaveable { mutableIntStateOf(-1) }
    val onItemClicked: (Int) -> Unit = { id ->
        currentIndex = id
        coroutineScope.launch { navigator.navigateTo(
                pane = ListDetailPaneScaffoldRole.Detail, contentKey = id
            )}}
    val detailVisible =
        navigator.scaffoldValue[ListDetailPaneScaffoldRole.Detail] == PaneAdaptedValue.Expanded
    if (detailVisible && currentIndex == -1) currentIndex = 0
    NavigationHelper(navigator = navigator,
        navigationState = navigationState,
        coroutineScope = coroutineScope)
    ListDetailPaneScaffold(
        directive = navigator.scaffoldDirective,
        value = navigator.scaffoldValue,
        listPane = { MyList(
                onItemClicked = onItemClicked,
                currentIndex = currentIndex,
                detailVisible = detailVisible
            )},
        detailPane = { MyListDetail(
                currentIndex = currentIndex,
                listHidden = navigator.scaffoldValue[ListDetailPaneScaffoldRole.List] == PaneAdaptedValue.Hidden)})
}
```

```kotlin
@Composable
fun HomePane(navigationState: NavigationState) {
    val coroutineScope = rememberCoroutineScope()
    val navigator = rememberListDetailPaneScaffoldNavigator<Int>(
        scaffoldDirective = calculatePaneScaffoldDirectiveWithTwoPanesOnMediumWidth(
            currentWindowAdaptiveInfo()
        ))
    var currentIndex by rememberSaveable { mutableIntStateOf(-1) }
    val onItemClicked: (Int) -> Unit = { id ->
        currentIndex = id
        coroutineScope.launch { navigator.navigateTo(
                pane = ListDetailPaneScaffoldRole.Detail, contentKey = id
            )}}
    val detailVisible =
        navigator.scaffoldValue[ListDetailPaneScaffoldRole.Detail] == PaneAdaptedValue.Expanded
    if (detailVisible && currentIndex == -1) currentIndex = 0
    NavigationHelper(navigator = navigator,
        navigationState = navigationState,
        coroutineScope = coroutineScope)
    ListDetailPaneScaffold(
        directive = navigator.scaffoldDirective,
        value = navigator.scaffoldValue,
        listPane = { MyList(
                onItemClicked = onItemClicked,
                currentIndex = currentIndex,
                detailVisible = detailVisible
            )},
        detailPane = { MyListDetail(
                currentIndex = currentIndex,
                listHidden = navigator.scaffoldValue[ListDetailPaneScaffoldRole.List] == PaneAdaptedValue.Hidden)})
}
```

Defines how the scaffold should arrange its panes

Indicates how each pane of the scaffold is adapted

SnappMobile

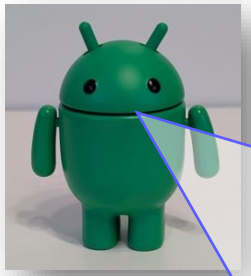
```kotlin
@Composable
fun ListDetailPaneScaffold(
    directive: PaneScaffoldDirective,
    value: ThreePaneScaffoldValue,
    listPane: @Composable ThreePaneScaffoldPaneScope.() -> Unit,
    detailPane: @Composable ThreePaneScaffoldPaneScope.() -> Unit,
    modifier: Modifier = Modifier,
    extraPane: (@Composable ThreePaneScaffoldPaneScope.() -> Unit)? = null,
    paneExpansionDragHandle: (@Composable ThreePaneScaffoldScope.(PaneExpansionState) -> Unit)? =
        null,
    paneExpansionState: PaneExpansionState? = null,
) {
    val expansionState =
        paneExpansionState
            ?: rememberDefaultPaneExpansionState(
                keyProvider = { value },
                mutable = paneExpansionDragHandle != null,
            )
    ThreePaneScaffold(
        modifier = modifier.fillMaxSize(),
        scaffoldDirective = directive,
        scaffoldValue = value,
        paneOrder = ListDetailPaneScaffoldDefaults.PaneOrder,
        secondaryPane = listPane,
        tertiaryPane = extraPane,
        paneExpansionDragHandle = paneExpansionDragHandle,
        paneExpansionState = expansionState,
        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

```kotlin
/**
 * 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")
```

SnappMobile



Let’s recap:

`directive` defines how the scaffold should arrange its panes

`value` indicates how each pane of the scaffold is adapted

We obtained the values from `navigator`: `navigator.scaffoldDirective` and `navigator.scaffoldValue`

So, `ThreePaneScaffoldNavigator` does all the work for us

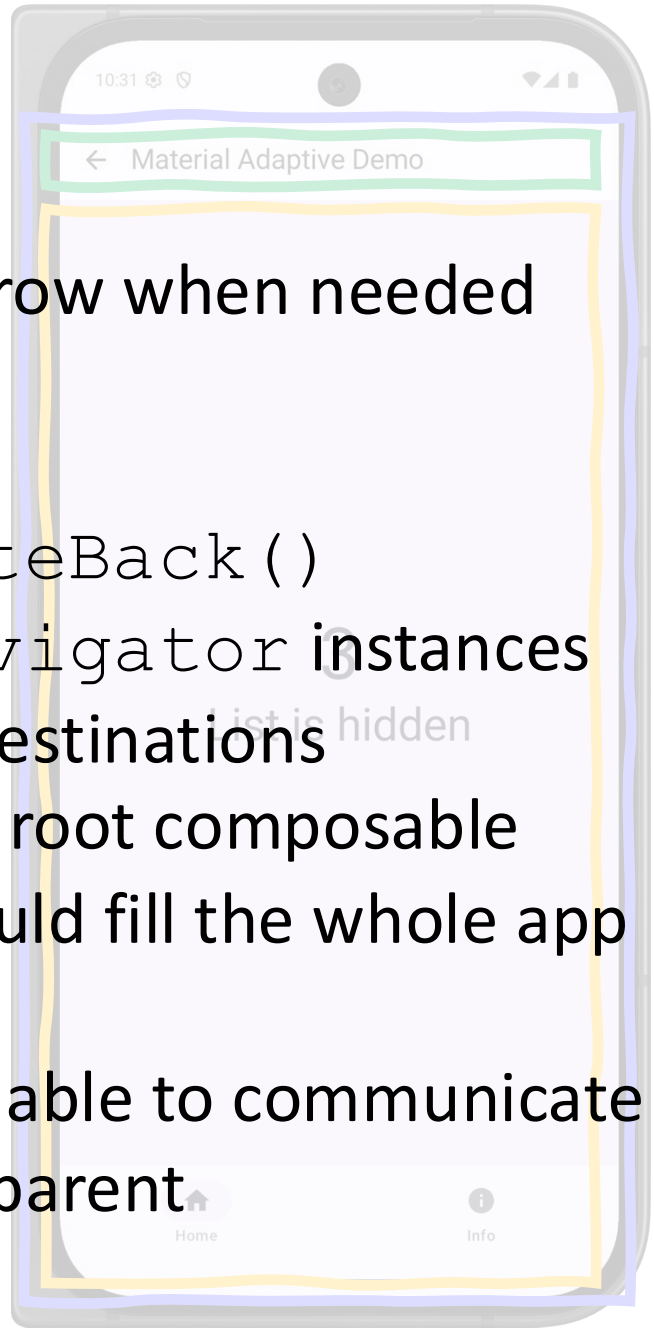
```kotlin
@Composable
fun <T> NavigationHelper(
    navigator: ThreePaneScaffoldNavigator<T>,
    navigationState: NavigationState,
    coroutineScope: CoroutineScope
) {
    LaunchedEffect(navigator.canNavigateBack()) {
        navigationState.update(
            canNavigateBack =
                navigator.canNavigateBack(),
            navigateBack = {
                coroutineScope.launch {
                    navigator.navigateBack()
                }
            }
        )
    }
    DisposableEffect(Unit) {
        onDispose {
            navigationState.clear()
        }
    }
    BackHandler(navigator.canNavigateBack()) {
        coroutineScope.launch {
            navigator.navigateBack()
        }
    }
}
```

```kotlin
class NavigationState {
    var canNavigateBack by mutableStateOf(false)
        private set

    var navigateBack: () -> Unit by mutableStateOf({})
        private set

    fun update(canNavigateBack: Boolean, navigateBack: () -> Unit) {
        this.canNavigateBack = canNavigateBack
        this.navigateBack = navigateBack
    }

    fun clear() {
        this.canNavigateBack = false
        this.navigateBack = {}
    }
}
```

That doesn’t look too complicated

But why do we need it at all?

- Top app bar shows a *Back* arrow when needed (pane-base navigation)
- We would use `navigator.canNavigateBack()`
- `ThreePaneScaffoldNavigator` instances are private to the top-level destinations
- But `Scaffold` must be the root composable because the top app bar should fill the whole app window width
- Therefore, children must be able to communicate their navigation state to the parent

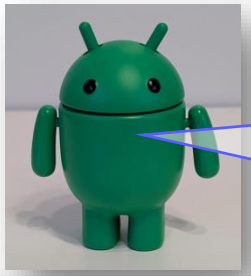



```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MaterialAdaptiveDemoScaffold() {
    var currentDestination by rememberSaveable { mutableStateOf(AppDestinations.Home) }
    val adaptiveInfo = currentWindowAdaptiveInfo()
    val customNavSuiteType =
        if (adaptiveInfo.windowSizeClass.isWidthAtLeastBreakpoint(WIDTH_DP_EXPANDED_LOWER_BOUND)) {
            NavigationSuiteType.NavigationDrawer
        } else {
            NavigationSuiteScaffoldDefaults.calculateFromAdaptiveInfo(adaptiveInfo)
        }
    val navigationState = remember { NavigationState() }
    Scaffold(
        contentWindowInsets = WindowInsets(),
        topBar = {
            TopAppBar(
                title = { Text(stringResource(Res.string.app_name)) },
                navigationIcon = {
                    if (navigationState.canNavigateBack) {
                        IconButton(
                            onClick = {
                                navigationState.navigateBack()
                            }
                        ) {
                            Icon(
                                Icons.AutoMirrored.Outlined.ArrowBack,
                                contentDescription = stringResource(Res.string.back)
                            )
                        }
                    }
                }
            )
        }) { paddingValues ->
        NavigationSuiteScaffold(
            modifier = Modifier.padding(paddingValues),
            layoutType = customNavSuiteType,
            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(navigationState)
                }

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

```kotlin
@Composable
fun ThreePaneScaffoldPaneScope.MyList(
    onItemClicked: (Int) -> Unit, currentIndex: Int, detailVisible: Boolean
) {
    AnimatedPane {
        LazyColumn {
            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()
                        }
                    }
                )
            }
        }
    }
}
```

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

SnappMobile



Let’s focus a little more on this one:

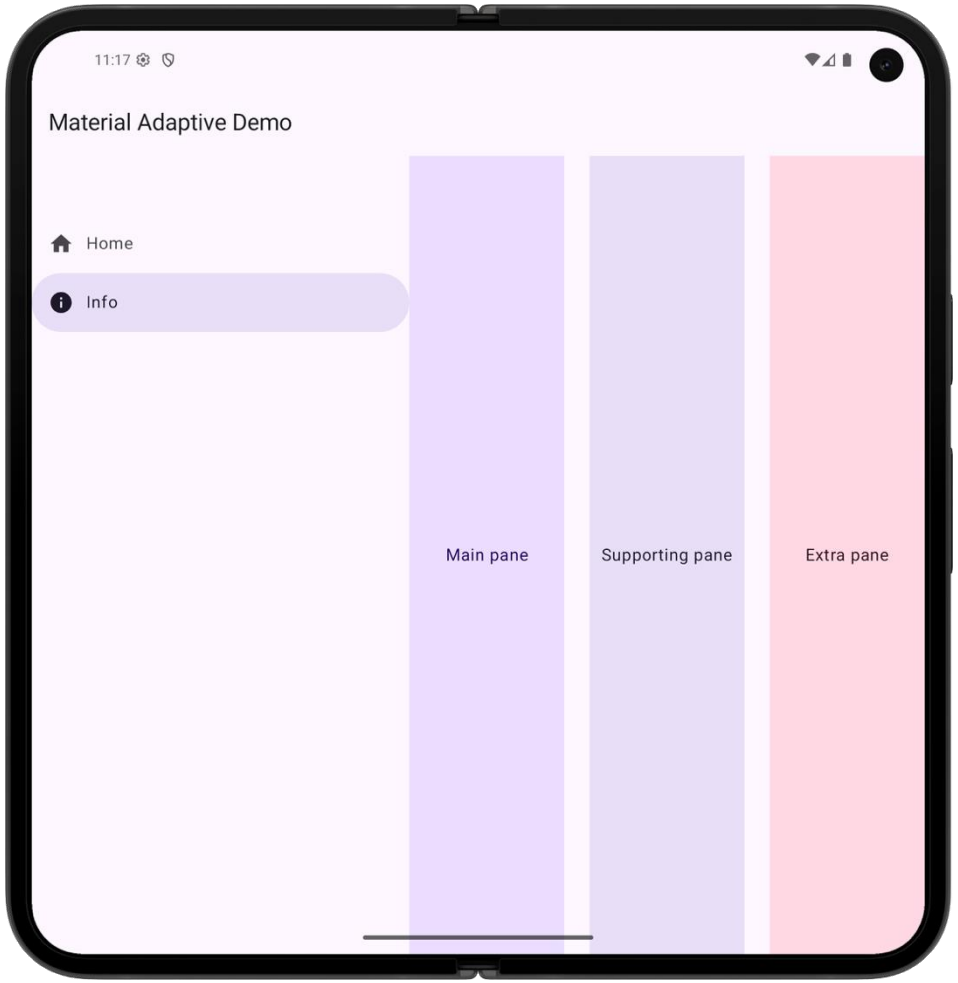
```
navigator.scaffoldValue[ListDetailPaneScaffoldRole.Detail] == PaneAdaptedValue.Expanded
```

```
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
    besides the list and the detail panes, for example, a task list or a m
    It maps to ThreePaneScaffoldRole.Tertiary.

    val Extra = ThreePaneScaffoldRole.Tertiary
}
```

```
    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")
```

Easily check if a pane is currently visible

```kotlin
@Composable
fun ThreePaneScaffoldPaneScope.MyListDetail(
    currentIndex: Int, listHidden: Boolean navigator.scaffoldValue[ListDetailPaneScaffoldRole.List] == PaneAdaptedValue.Hidden,
) {
    AnimatedPane {
        Column(
            horizontalAlignment = Alignment.CenterHorizontally,
            verticalArrangement = Arrangement.Center,
            modifier = Modifier.background(MaterialTheme.colorScheme.secondaryContainer)
        ) {
            Text(
                "${currentIndex + 1}",
                style = MaterialTheme.typography.displayLarge,
                color = MaterialTheme.colorScheme.onSecondaryContainer
            )
            if (listHidden) {
                Text(
                    text = stringResource(Res.string.list_hidden),
                    style = MaterialTheme.typography.displaySmall,
                    color = MaterialTheme.colorScheme.onSecondaryContainer
                )
            }
        }
    }
}
```



Example of a supporting pane

```kotlin
@Composable
fun InfoPane(navigationState: NavigationState) {
    val scope = rememberCoroutineScope()
    val navigator = rememberSupportingPaneScaffoldNavigator(
        scaffoldDirective = calculatePaneScaffoldDirective(currentWindowAdaptiveInfo()).copy(
            maxHorizontalPartitions = with(currentWindowAdaptiveInfo().windowSizeClass) {
                if (isWidthAtLeastBreakpoint(WIDTH_DP_EXPANDED_LOWER_BOUND))
                    3
                else if (isWidthAtLeastBreakpoint(WIDTH_DP_MEDIUM_LOWER_BOUND))
                    2
                else 1
            }
        )
    )
    NavigationHelper(
        navigator = navigator,
        navigationState = navigationState,
        coroutineScope = scope
    )
    SupportingPaneScaffold(
        directive = navigator.scaffoldDirective,
        mainPane = { MainPane(navigator = navigator) },
        supportingPane = { SupportingPane(navigator = navigator) },
        extraPane = { ExtraPane() },
        value = navigator.scaffoldValue
    )
}
```

```kotlin
@Composable
fun SupportingPaneScaffold(
    directive: PaneScaffoldDirective,
    value: ThreePaneScaffoldValue,
    mainPane: @Composable ThreePaneScaffoldPaneScope.() -> Unit,
    supportingPane: @Composable ThreePaneScaffoldPaneScope.() -> Unit,
    modifier: Modifier = Modifier,
    extraPane: (@Composable ThreePaneScaffoldPaneScope.() -> Unit)? = null,
    paneExpansionDragHandle: (@Composable ThreePaneScaffoldScope.(PaneExpansionState) -> Unit)? =
        null,
    paneExpansionState: PaneExpansionState? = null,
) {
    val expansionState =
        paneExpansionState
            ?: rememberDefaultPaneExpansionState(
                keyProvider = { value },
                mutable = paneExpansionDragHandle != null,
            )
    ThreePaneScaffold(
        modifier = modifier.fillMaxSize(),
        scaffoldDirective = directive,
        scaffoldValue = value,
        paneOrder = SupportingPaneScaffoldDefaults.PaneOrder,
        secondaryPane = supportingPane,
        tertiaryPane = extraPane,
        paneExpansionDragHandle = paneExpansionDragHandle,
        paneExpansionState = expansionState,
        primaryPane = mainPane,
    )
}
```

```kotlin
@Composable
fun ThreePaneScaffoldPaneScope.MainPane(navigator: ThreePaneScaffoldNavigator<Any>) {
    val scope = rememberCoroutineScope()
    ColoredBox(
        textColor = MaterialTheme.colorScheme.onPrimaryContainer,
        backgroundColor = MaterialTheme.colorScheme.primaryContainer,
        resIdMessage = Res.string.main_pane,
        resIdButton = Res.string.show_supporting_pane,
        shouldShowButton = navigator.scaffoldValue[SupportingPaneScaffoldRole.Supporting] == PaneAdaptedValue.Hidden,
        onClick = { scope.launch { navigator.navigateTo(SupportingPaneScaffoldRole.Supporting) } })
}
```

```kotlin
@Composable
fun ThreePaneScaffoldPaneScope.ColoredBox(
    textColor: Color, backgroundColor: Color, shouldShowButton: Boolean,
    resIdMessage: StringResource, resIdButton: StringResource,
    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(stringResource(resIdButton)) }
            }
        }
    }
}
```

```kotlin
@Composable
fun ThreePaneScaffoldPaneScope.SupportingPane(navigator: ThreePaneScaffoldNavigator<Any>) {
    val scope = rememberCoroutineScope()
    ColoredBox(
        textColor = MaterialTheme.colorScheme.onSecondaryContainer,
        backgroundColor = MaterialTheme.colorScheme.secondaryContainer,
        resIdMessage = Res.string.supporting_pane,
        resIdButton = Res.string.show_main_pane,
        shouldShowButton = navigator.scaffoldValue[SupportingPaneScaffoldRole.Main] == PaneAdaptedValue.Hidden,
        onClick = { scope.launch { navigator.navigateBack() } })
}
```

```kotlin
@Composable
fun ThreePaneScaffoldPaneScope.ExtraPane() {
    ColoredBox(
        textColor = MaterialTheme.colorScheme.onTertiaryContainer,
        backgroundColor = MaterialTheme.colorScheme.tertiaryContainer,
        resIdMessage = Res.string.extra_pane,
        resIdButton = Res.string.main_pane,
        shouldShowButton = false
    )
}
```

One more thing

- Material 3 Adaptive shows or hides panes based on Window Size classes
- Provides means to navigate between panes
- *List-detail* implemented by `ListDetailPaneScaffold` and `NavigableListDetailPaneScaffold`
- *Supporting Pane* implemented by `SupportingPaneScaffold` and `NavigableSupportingPaneScaffold`
- No implementation for *Feed*

```kotlin
@ExperimentalMaterial3AdaptiveApi
@Composable
fun <T> NavigableListDetailPaneScaffold(
    navigator: ThreePaneScaffoldNavigator<T>,
    listPane: @Composable ThreePaneScaffoldPaneScope.() -> Unit,
    detailPane: @Composable ThreePaneScaffoldPaneScope.() -> Unit,
    modifier: Modifier = Modifier,
    extraPane: (@Composable ThreePaneScaffoldPaneScope.() -> Unit)? = null,
    defaultBackBehavior: BackNavigationBehavior =
        BackNavigationBehavior.PopUntilScaffoldValueChange,
    paneExpansionDragHandle: (@Composable ThreePaneScaffoldScope.(PaneExpansionState) -> Unit)? =
        null,
    paneExpansionState: PaneExpansionState? = null,
) {
    ThreePaneScaffoldPredictiveBackHandler(
        navigator = navigator,
        backBehavior = defaultBackBehavior,
    )

    ListDetailPaneScaffold(
        modifier = modifier,
        directive = navigator.scaffoldDirective,
        scaffoldState = navigator.scaffoldState,
        detailPane = detailPane,
        listPane = listPane,
        extraPane = extraPane,
        paneExpansionDragHandle = paneExpansionDragHandle,
        paneExpansionState = paneExpansionState,
    )
}
```

An effect to add predictive back handling to a three pane scaffold.

```kotlin
@Composable
fun <T> NavigationHelper(
    navigator: ThreePaneScaffoldNavigator<T>,
    navigationState: NavigationState,
    coroutineScope: CoroutineScope
) {
    LaunchedEffect(navigator.canNavigateBack()) {
        navigationState.update(
            canNavigateBack =
                navigator.canNavigateBack(),
            navigateBack = {
                coroutineScope.launch {
                    navigator.navigateBack()
                }
            }
        )
    }
    DisposableEffect(Unit) {
        onDispose {
            navigationState.clear()
        }
    }
    BackHandler(navigator.canNavigateBack()) {
        coroutineScope.launch {
            navigator.navigateBack()
        }
    }
}
```

```kotlin
package de.thomaskuenneth.material.adaptive.multiplatform

import androidx.compose.runtime.Composable

@Composable
expect fun BackHandler(enabled: Boolean = true, onBack: () -> Unit)
```

```kotlin
package de.thomaskuenneth.material.adaptive.multiplatform

import androidx.activity.compose.BackHandler
import androidx.compose.runtime.Composable

@Composable
actual fun BackHandler(enabled: Boolean, onBack: () -> Unit) {
    BackHandler(enabled = enabled, onBack = onBack)
}
```

```kotlin
package de.thomaskuenneth.material.adaptive.multiplatform

import androidx.compose.runtime.Composable

@Composable
actual fun BackHandler(enabled: Boolean, onBack: () -> Unit) {
}
```

Finishing line

- Supporting different form factors used to be a pain on Android
- Google neglected tablets for way too long
- The rise of foldables provided a second chance
- Google took it, finally investing in adaptive UIs (Material 3, Window Size Classes, Jetpack Window Manager)

- Material 3 provides the adaptive theory
- Jetpack libraries provide the implementation
- Compose Multiplatform relies on Jetpack Compose, so it benefits from (almost) all of this

- Advanced integrations still need platform-specific code (menu bars, multiple windows, etc.)
- But, building responsive, shared-codebase apps for Android, iOS, and Desktop is much easier with Compose Multiplatform

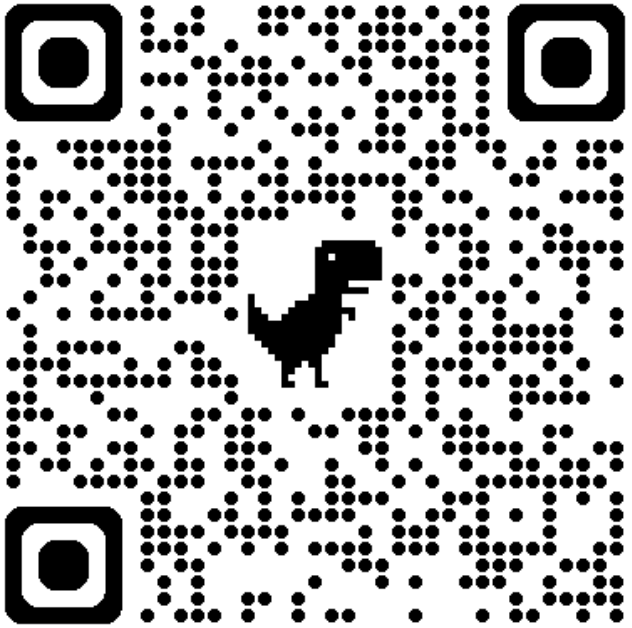
Thank you!

@tkuenneth

@tkuenneth

@tkuenneth.dev

@tkuenneth@mastodon.social



https://github.com/tkuenneth/foldables_and_large_screens

Experts

SnappMobile