CMPS 312

Navigation

Dr. Abdelkarim Erradi

CSE@QU

Navigation

The act of moving between screens of an app to complete tasks

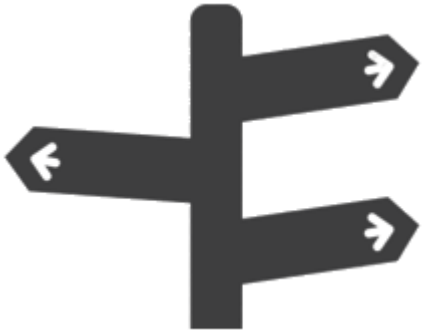
Designing effective navigation = Simplify the user journey

Outline

1. Navigation
2. Navigation Widgets
3. Responsive Navigation UI
4. Floating Windows

Navigation

Used for navigating between destinations within an app



Back

https://docs.flutter.dev/ui/navigation

GoRouter

- First, add GoRouter to your pubspec.yaml. Then, configure the routes and integrate the router with the MaterialApp

```dart
// 1. Define your routes
final _router = GoRouter(
  initialLocation: '/home', // The path to show on app launch
  routes: [
    GoRoute(
      path: '/home',
      builder: (context, state) => HomeScreen(),
    ),
    GoRoute(
      path: '/details',
      builder: (context, state) => DetailsScreen(),
    ),
  ],
);
```

```dart
// 2. Integrate with MaterialApp
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp.router(
      routerConfig: _router,
      title: 'GoRouter Example',
    );
  }
}
```

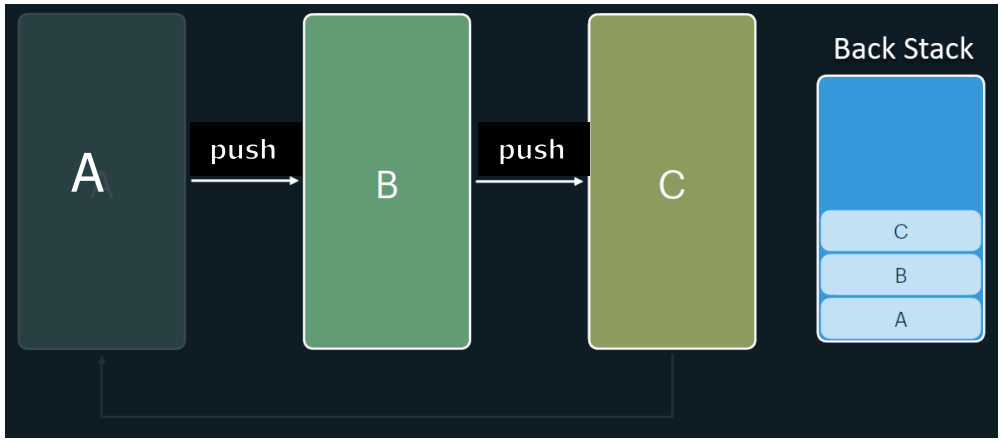
Navigating Between Screens

- You can navigate using extension methods on the BuildContext:
  - **`context.go()`**: Navigates to a new screen, replacing the current route. Good for destinations reached from a BottomNavigationBar or Navigating after successful login
  - **`context.push()`**: Pushes a new screen onto the top of the navigation stack. The user can press the back button to return to the previous screen

```
// In your HomeScreen widget
ElevatedButton(
  // Navigates to the details screen and allows returning
  onPressed: () => context.push('/details'),
  child: const Text('View Details'),
)
```

Navigation and Back Stack Control

- **push()** - Add a Route on top of the Stack for displaying new screen
  - Router keeps track of the **back stack** of visited screens
  - Perfect for **detail screens, forms, dialogs**, or any drill-down flow
  - E.g., From home, push product details



context.push('/product/123');

- **pop()** - removes the current route, returning to the previous one

```
// Inside a details screen
IconButton(
  icon: const Icon(Icons.arrow_back),
  onPressed: () =>
    if (context.canPop()) {
      context.pop();
    } else {
      // Already at root – maybe exit app or
         show dialog
    },
);
```

Passing Parameters Between Screens

- Path Parameters:
  - Use path parameters for simple, required identifiers like a product ID. They are part of the URL itself
  - Scenario: Navigating from a list of products to a specific product's detail page

```
// In GoRouter configuration
GoRoute(
  // The ':id' part is a path parameter
  path: '/product/:id',
  builder: (context, state) {
    // Extract the parameter
    final productId = state.pathParameters['id']!;
    return ProductDetailScreen(productId: productId);
  },
),
```

```
// In your product list screen
ListTile(
  title: const Text('Awesome Gadget'),
  onTap: () => context.push('/product/123')
)
```

Query Parameters

- Use query parameters for optional or filtering data, similar to how they are used on the web
- Scenario: A search screen where the search term and filters are passed in the URL

```dart
// In GoRouter configuration
GoRoute(
  path: '/search',
  builder: (context, state) {
    // Extract query parameters
    final searchTerm = state.uri.queryParameters['q']; // Using state.uri is safer
    final sortBy = state.uri.queryParameters['sortBy'] ?? 'relevance';
    return SearchResultsScreen(searchTerm: searchTerm, sortBy: sortBy);
  },
),
```

```dart
// In your search bar widget
void _onSearchSubmitted(String term) {
  // Navigates to a URL like: /search?q=flutter&sortBy=date
  context.go('/search?q=$term&sortBy=date');
}
```

Passing Objects

- Example: Tapping on a User object in a list and passing the entire object to the profile screen to avoid re-fetching the data

```
// In GoRouter configuration
GoRoute(
  path: '/profile',
  builder: (context, state) {
    // Extract the object from the 'extra' field
    final user = state.extra as User; // Cast to your object type
    return ProfileScreen(user: user);
  },
),
```

```
// In your user list screen
final user = User(id: '456', name: 'Jane Doe');
ListTile(
  title: Text(user.name),
  onTap: () => context.push('/profile', extra: user), // Pass the whole object
)
```

Navigation Widgets:

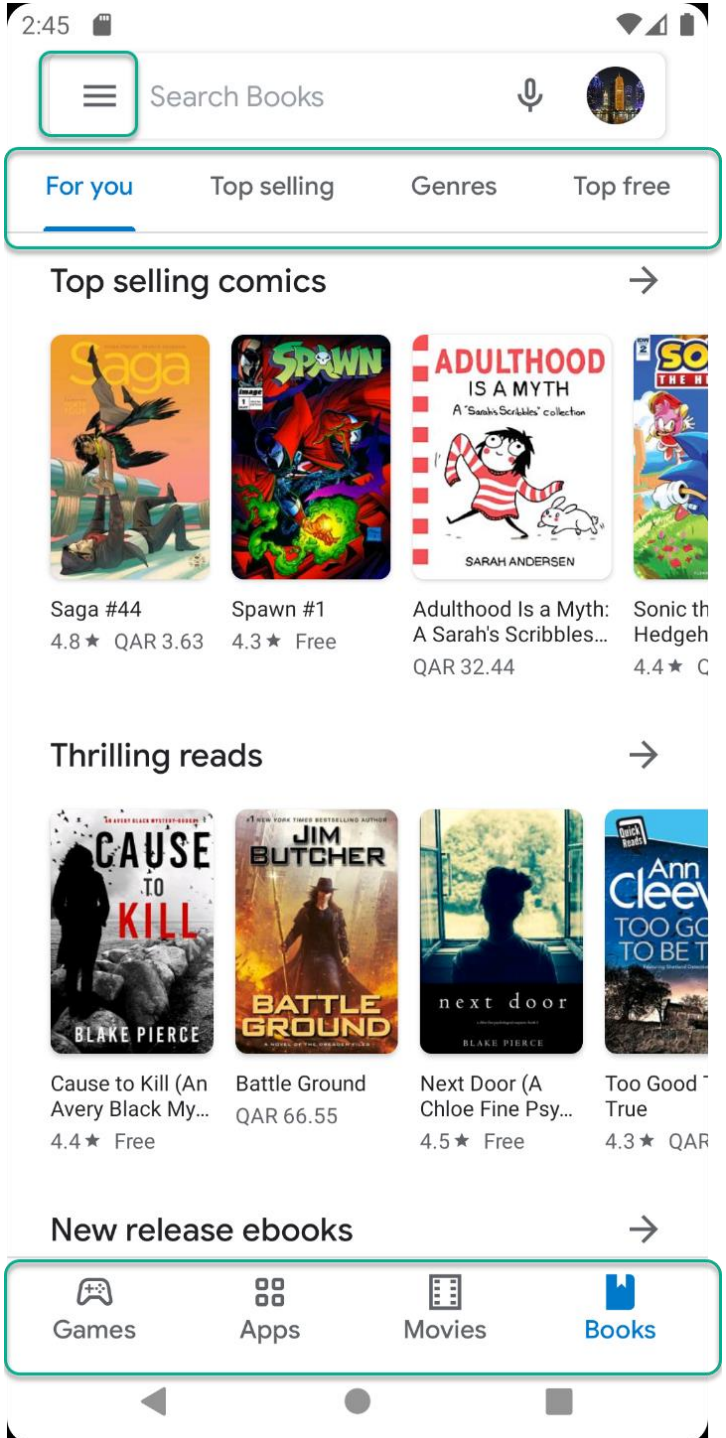
App Bars

BottomNavigationBar

Navigation Rail

Floating Action Button

Navigation Drawer

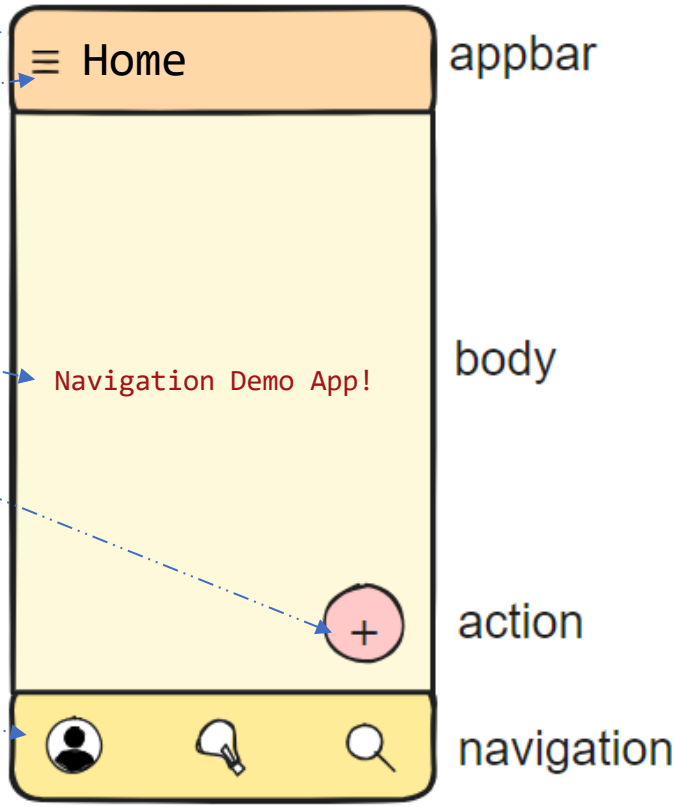


Back

Scaffold

- **Scaffold** is a **Slot-based** layout
- Scaffold is **template** to build the entire screen by adding different UI Navigation components (e.g., ***appBar***, ***bottomNavigationBar***, **floatingActionButton**, ***drawer***)
- The main content is assigned to the **body** property

```
Scaffold(
    appBar: AppBar(
      title: const Text('Home'),
    ),
    drawer: const NavDrawer(),
    body: const Center(
      child: Text('Navigation Demo App!'),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: () {
        Navigator.pushNamed(context, 'fruits');
      },
      child: const Icon(Icons.local_grocery_store),
    ),
    bottomNavigationBar: BottomNavBar(
      selectedIndex: _selectedIndex,
      onTapNavItem: _onTapNavItem,
    ),
)
```



AppBar

- Info and actions **related to the current screen**
- Typically has Title, Drawer button / Back button, Menu items

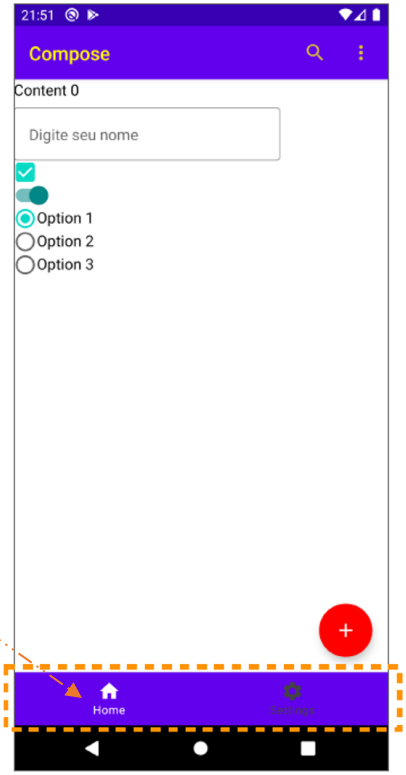
```
TopAppBar(
    title = {
        Text(text = "Compose")
    },
    navigationIcon = {
        IconButton(onClick = { }) {
            Icon(
                imageVector = Icons.Default.Search,
                contentDescription = "Search"
            )
        }
    }

    navigationIcon = {
        IconButton(onClick = { }) {
            Icon(
                imageVector = Icons.Default.MoreVert,
                contentDescription = "More"
            )
        }
    }
)
```

Bottom Navigation Bar

- Allow movement between the app's primary **top-level destinations** (3 to 5 options)
- Each destination is represented by an icon and an optional text label. May have notification badges

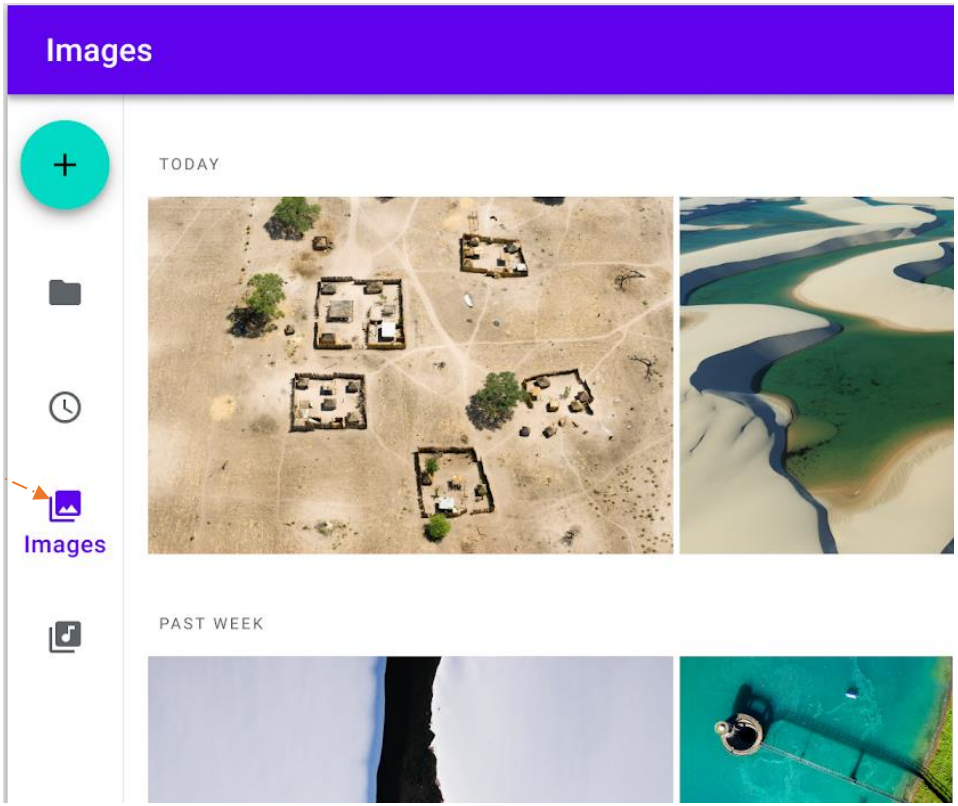

```
NavigationBar {
   NavigationBarItem(
     icon = {Icon(Icons.Default.Home,
                  contentDescription = "Home")},
     label = { Text( "Home" ) }
     onClick = { },
   )
   ...
   NavigationBarItem(
     icon = { },
     label = { }
     onClick = { },
   )
}
```



Navigation Rail

- Can contain 3-7 destinations plus an optional FAB
- Recommended for for **medium** or **expanded** screens

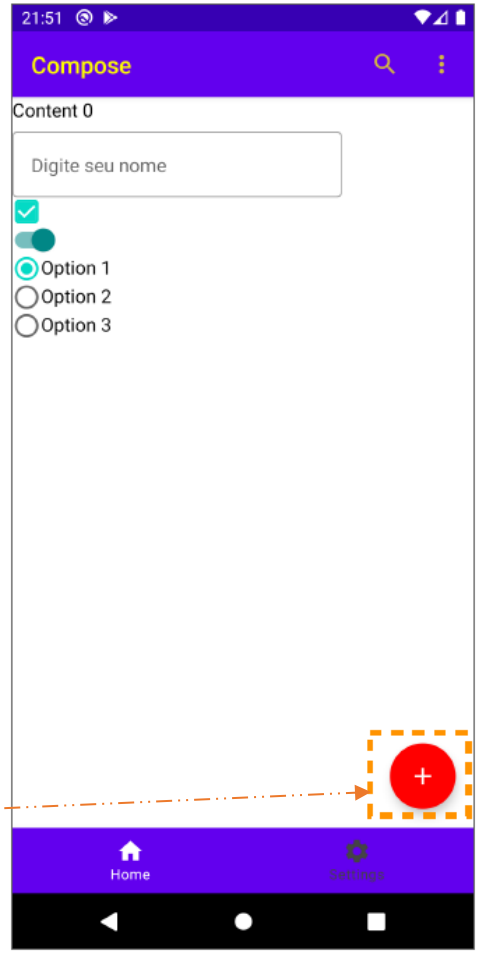
```
NavigationRail {
    ...
    NavigationRailItem(
      icon = {Icon(Icons.Default.Image,
             contentDescription = "Images")},
      label = { Text( "Images" ) }
      onClick = { },
    )
    ...
    NavigationRailItem(
      icon = { },
      label = { }
      onClick = { },
    )
}
```



Floating Action Button (FAB)

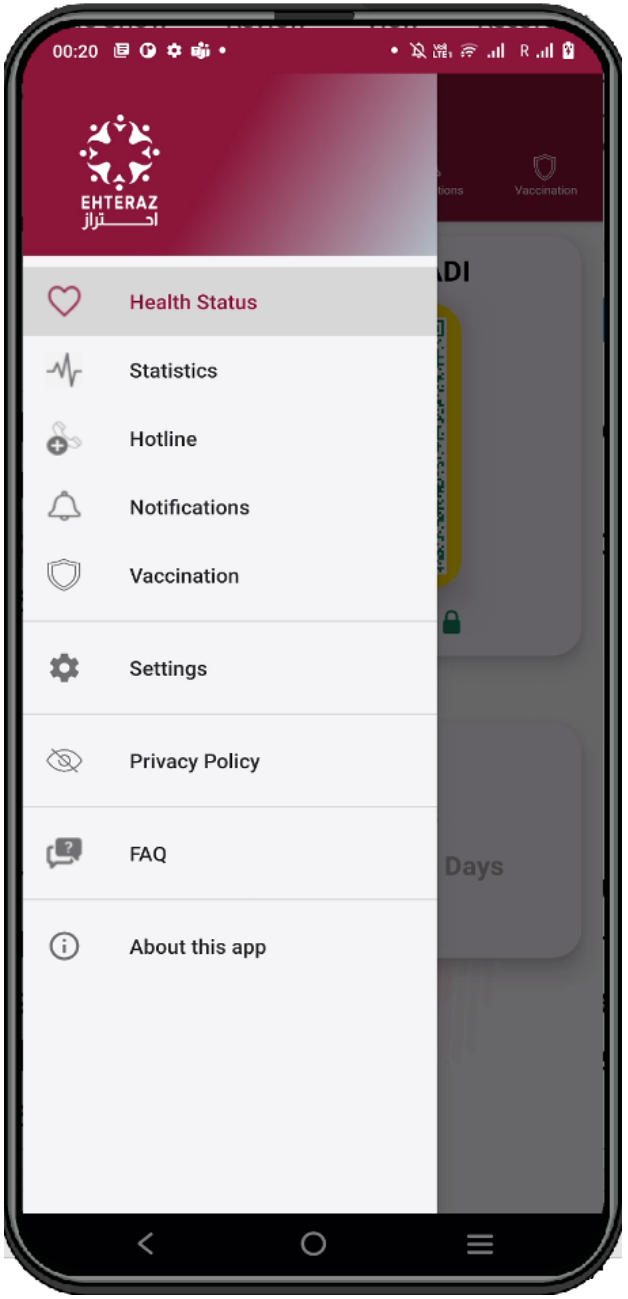
- A FAB performs the primary, or most common, action on a screen, such as drafting a new email
  - It appears in front of all screen content, typically as a circular shape with an icon in its center.
  - FAB is typically placed at the bottom right

```
FloatingActionButton(
    onClick = { … },
    backgroundColor = Color.Red,
    contentColor = Color.White
) {
    Icon(Icons.Filled.Add, "Add")
}
```



Navigation Drawer

- Navigation Drawer provides access to app **destinations** that cannot fit on the Bottom Bar , such as settings screen
  - Recommended for five or more top-level destinations
  - Quick navigation between unrelated destinations
- The drawer appears when the user touches the drawer icon ☰ in the app bar or when the user swipes a finger from the left edge of the screen

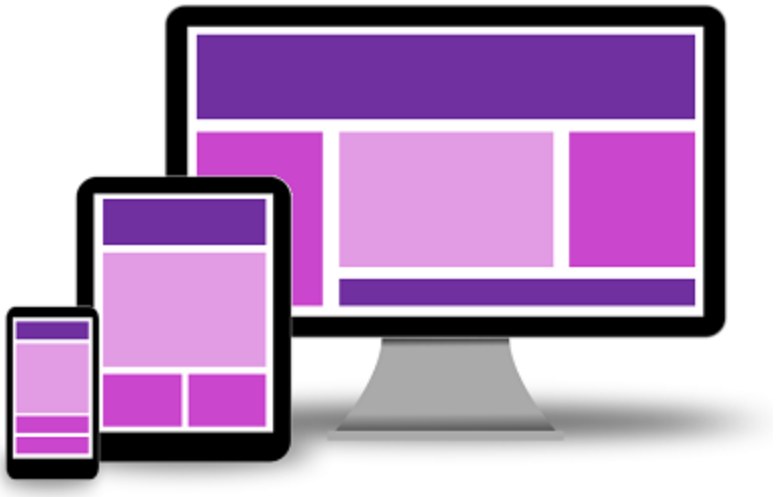


Navigation Drawer - Example

```
Drawer(
    drawerContent = {
        ModalDrawerSheet {
            NavigationDrawerItem(
               label = { Text(text = "Settings" ) },
               icon = { Icon(Icons.Default.Settings,
                                  contentDescription = "Settings")
                      },
               onClick = { }
            )
           ...
        }
})
```

- See more details in the posted example

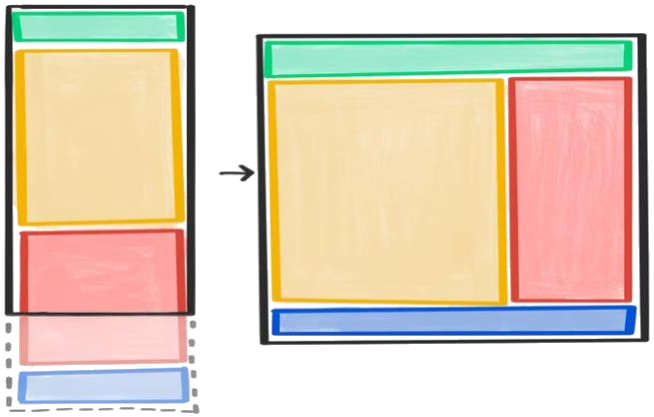
Responsive Navigation UI



Back

Responsive UI

- Responsive UI = **serve different layouts for different screen sizes and orientations**
  - **Optimize the viewing experience on range of devices**: mobile, desktop, tablet, TV…
- For example, a newspaper app might have a single column of text on a mobile device, but display several columns on a larger tablet/desktop device



MediaQuery.of(context).size

- `MediaQuery.of(context).size` return the window size class

```
val screenSize =
MediaQuery.of(context).size
```

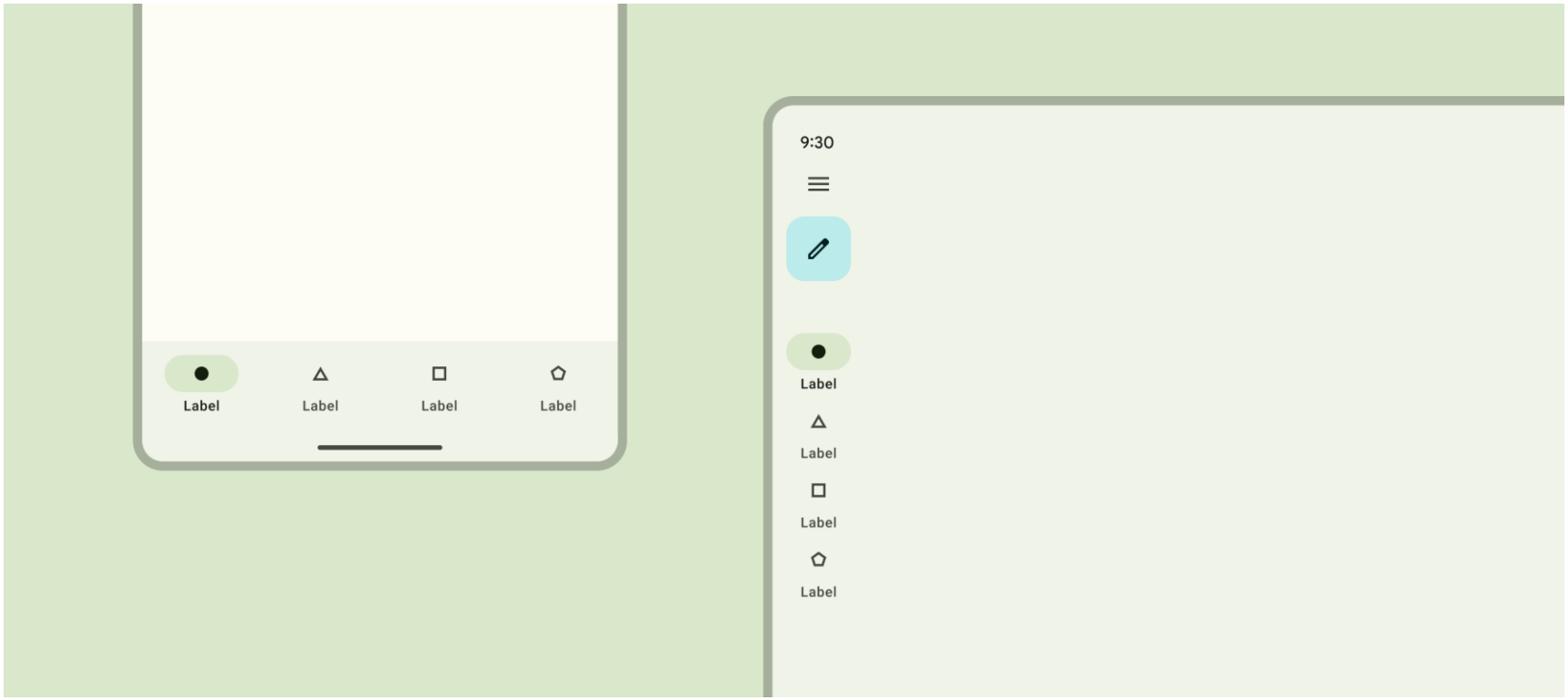
Design for window size classes instead of specific devices

- Devices fall into different window size classes based on orientation and user behavior, such as multi-window modes or unfolding a foldable device
- Start by designing for compact window class size and then adjust your layout for the next class size

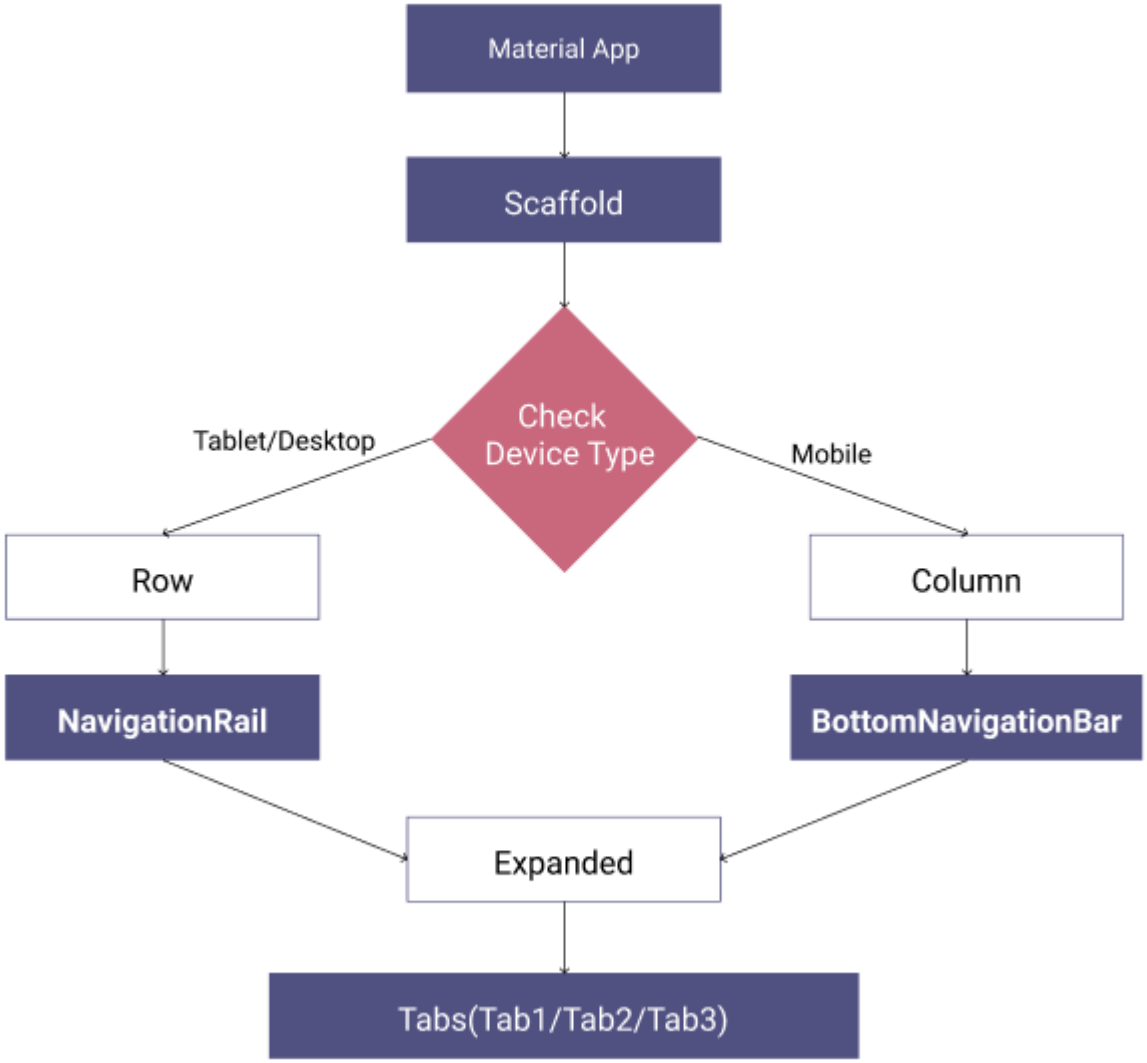
Window class (width)	Breakpoint (dp)	Common devices
Compact	Width < 600	Phone in portrait
Medium	600 <= width < 840	Tablet in portrait Foldable in portrait (unfolded)
Expanded	Width >= 840	Phone in landscape Tablet in landscape Foldable in landscape (unfolded) Desktop

Responsive UI - Example

- A bottom navigation bar in a compact layout can be swapped with a navigation rail in a medium layout, and a navigation drawer in an expanded layout



Responsive UI - Example



Responsive UI - Example

```
val context = LocalContext.current as Activity
val windowSizeClass = calculateWindowSizeClass(context)

val shouldShowBottomBar =  windowSizeClass.widthSizeClass
       == WindowWidthSizeClass.Compact
val shouldShowNavRail = !shouldShowBottomBar
...
Scaffold(
    bottomBar = {
        if (shouldShowBottomBar)
            BottomNavBar(navController)
    }
) {
    padding -> Row(...) {
        if (shouldShowNavRail) {
            AppNavigationRail(navController)
        }
        AppNavigator(navController = navController)
     }
}
```

Floating Windows

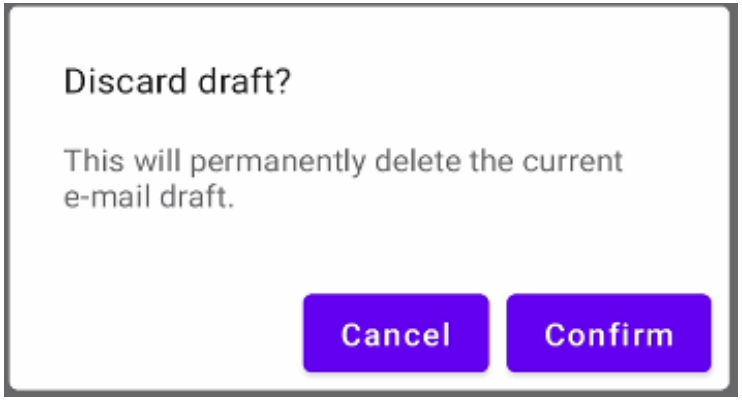
Back

Alert Dialog

- Alert dialog is a Dialog which interrupts the user with urgent information, details or actions
- Dialogs are displayed in front of app content
  - Inform users about a task that may contain **critical information** and/or **require a decision**
  - Interrupt the current flow and remain on screen until dismissed or action taken. Hence, they should be used sparingly
- 3 Common Usage:
  - **Alert dialog**: request user action/confirmation. Has a title, optional supporting text and action buttons
  - **Simple dialog:** Used to present the user with a list of actions that, when tapped, take immediate effect.
  - **Confirmation dialog:** Used to present a list of single- or multi-select choices to a user. Action buttons serve to confirm the choice(s)

Alert Dialog

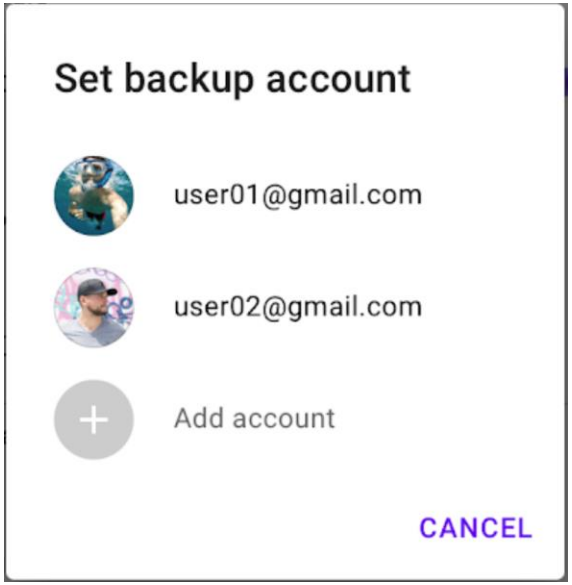
- Commonly used to **confirm high-risk actions** like deleting progress



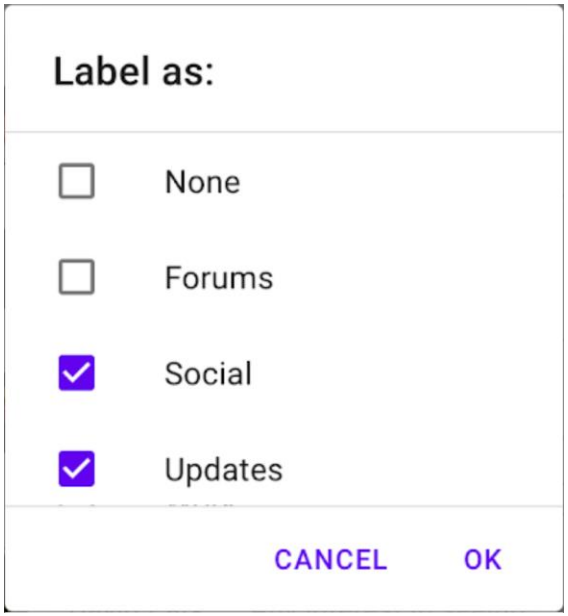

```
AlertDialog(
        onDismissRequest = {
           // Dismiss the dialog when the user clicks outside the dialog
           // or on the back button
             onDialogOpenChange(false)
        },
        title = { Text(text = title) },
        text = { Text(text = message) },
        confirmButton = {
            Button(
                onClick = {  onDialogResult(true) }) {
                Text(text = "Confirm")
            }
        },
        dismissButton = {
            Button(
                onClick = {  onDialogResult(false) }) {
                Text("Cancel")
            }
        }
    )
}
```

Simple dialog:

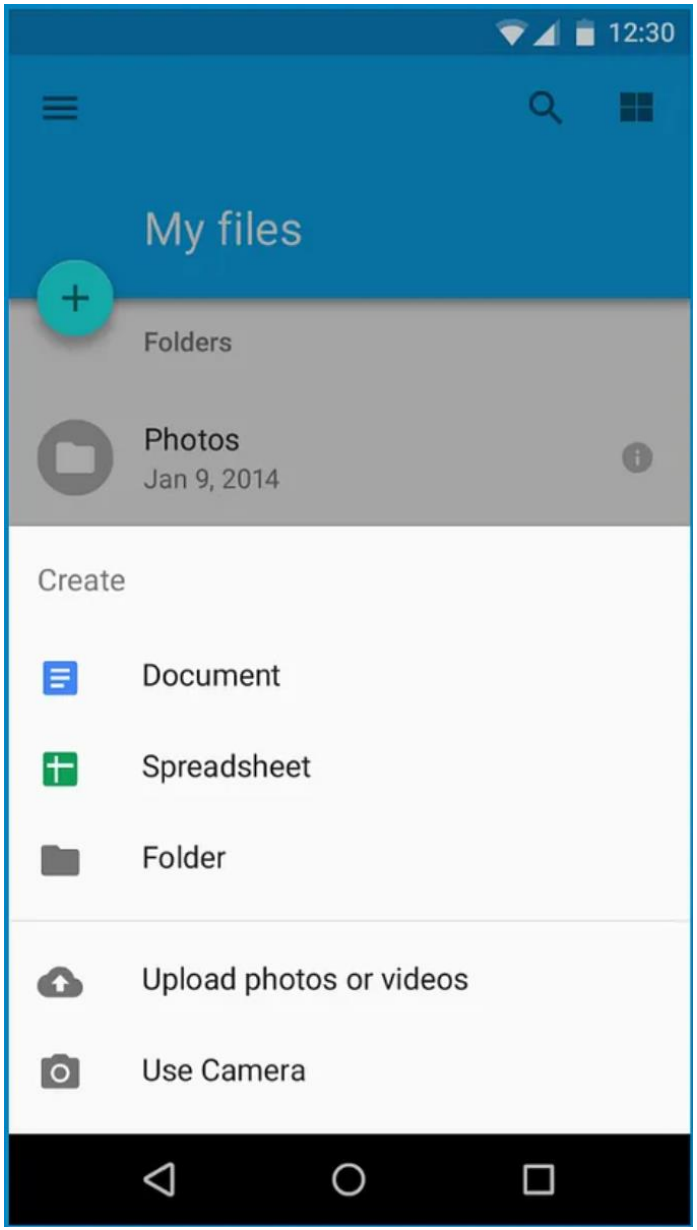
present the user with a list of actions that, when tapped, take immediate effect



Confirmation dialog (multi choice)



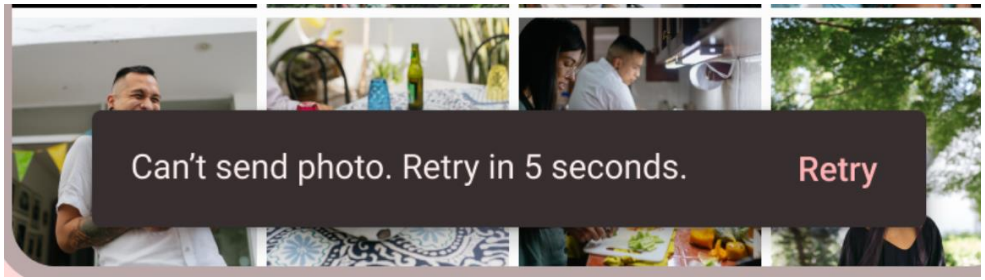
Bottom Sheets



- Bottom sheets show secondary content / actions anchored to the bottom of the screen
- Content should be additional or secondary (not the app's main content)
- Bottom sheets can be dismissed in order to interact with the main content
- See more details in the posted example

Snackbar

- Snackbars show **short updates** about app processes at the bottom of the screen



- Do not interrupt the user’s experience
- Can disappear on their own or remain on screen until the user takes action
- See more details in the posted example

Define a Destination Class to Enumerate the App Destinations

- Define a Destination class to enumerate the app destinations to shown in the example below

```
class Destination {
  const Destination(this.icon, this.label);
  final IconData icon;
  final String label;
}

const List<Destination> destinations = <Destination>[
  Destination(Icons.inbox_rounded, 'Inbox'),
  Destination(Icons.article_outlined, 'Articles'),
  Destination(Icons.messenger_outline_rounded, 'Messages'),
  Destination(Icons.group_outlined, 'Groups'),
];
```

Resources

- Declarative navigation using go_router package
- Flutter Navigation
  - https://docs.flutter.dev/ui/navigation
- Flutter Navigation hands-on practice
  - https://docs.flutter.dev/cookbook#navigation