CMPS 312

Navigation

Dr. Abdelkarim Erradi

CSE@QU

Navigation

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

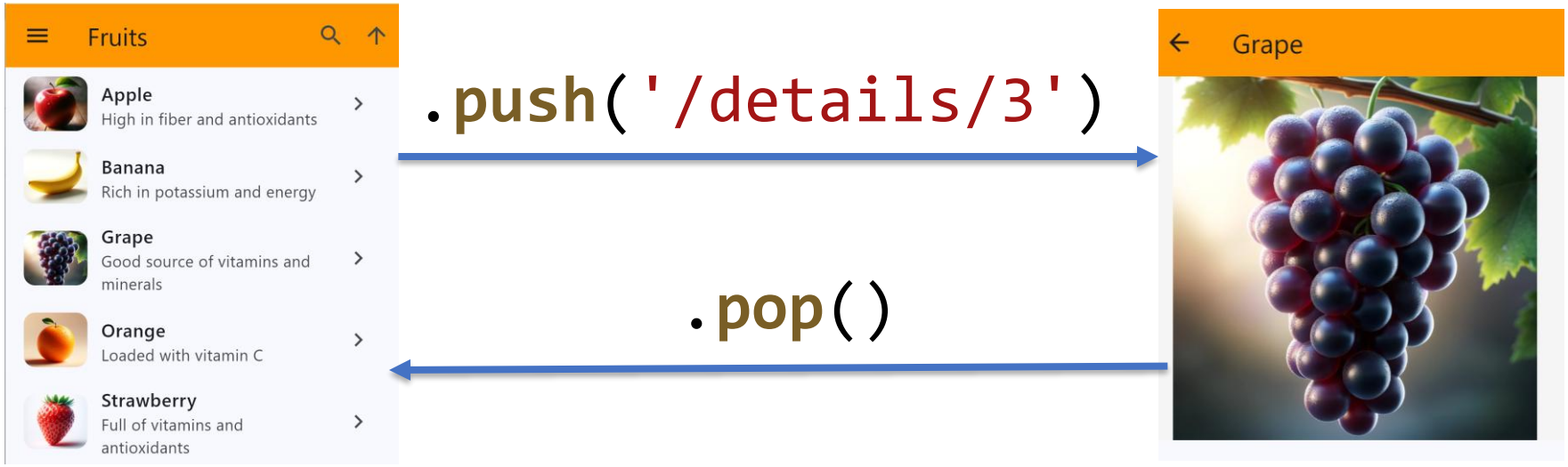
Design effective navigation to Simplify the user journey

Outline

1. Navigation
2. Navigation Widgets
3. Adaptive Navigation
4. Dialogs and Sheets

Navigation

Used for navigating between destinations within an app



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',
    );
  }
}
```

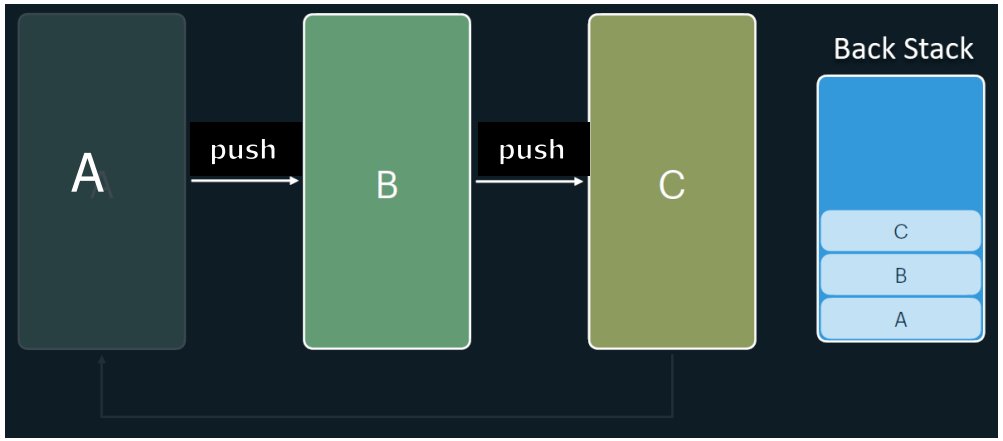
Navigating Between Screens

- 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 Back 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 products list use **.push** to navigate to 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 parameters 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

```
// 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);
  },
),
```

```
// 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);
  },
),
```

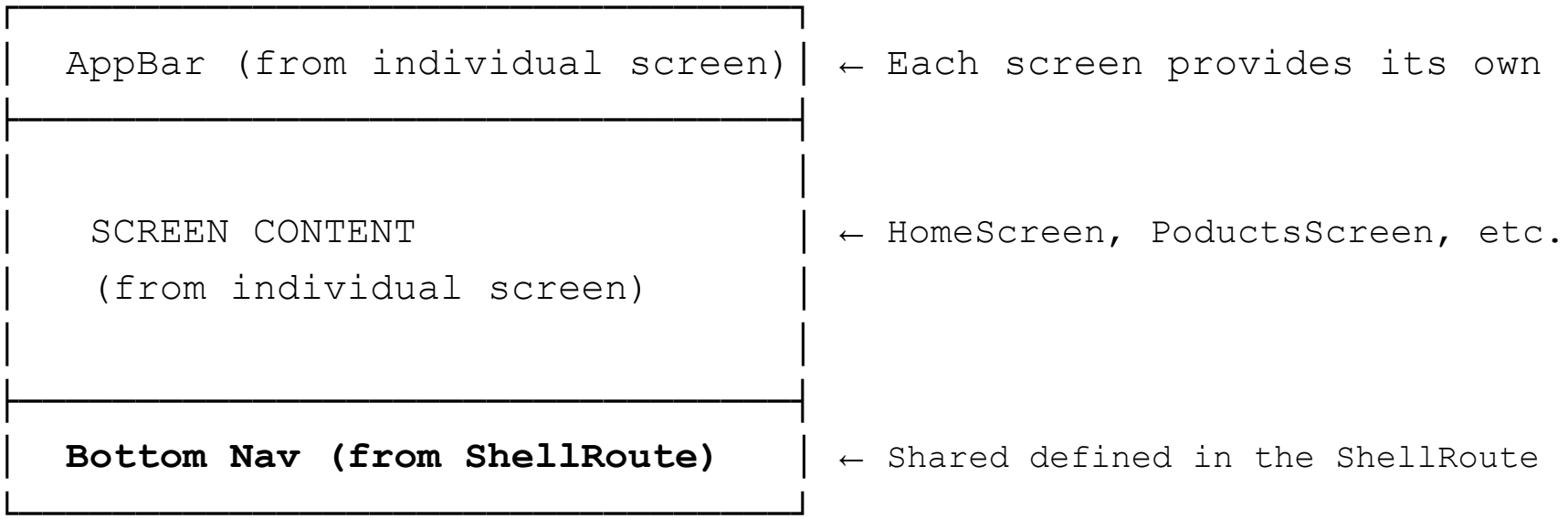
```
// In your user list screen
final user = User(id: '456', name: 'Jane Doe');
ListTile(
  title: Text(user.name),
  onTap: () => context.push('/profile', extra: user), // Pass the whole object
)
```

Navigation Key Methods

```
// Navigate to route
context.go('/fruits');

// Push with data
context.push('/details', extra: fruit);

// Pop
context.pop();

// Pop with result
context.pop(resultData);
```

What is ShellRoute?

- **ShellRoute** is a special route that wraps multiple child routes with a common UI shell (like a persistent bottom navigation bar)



How It Works

```
ShellRoute(
  builder: (context, state, child) {
    return Scaffold(
      body: child,  // ← Individual screen

      // Common persistent bottom navigation bar

      bottomNavigationBar: BottomNavBar(...),
    );
  },
  routes: [
    GoRoute(path: '/',

       builder: (context, state) => const HomeScreen()),

     GoRoute(path: '/products',

       builder: (context, state) => const ProductsScreen())
)
```

When to Use ShellRoute

✅ Use when you have:

- Common Nav UI such as Bottom navigation bar that persists across screen
  - Flexible Per-Screen Customization: each screen can define its own AppBar

❌ Don’t use for:

- Detail screens (put outside ShellRoute)
- Screens having completely different layouts

Navigation Widgets:

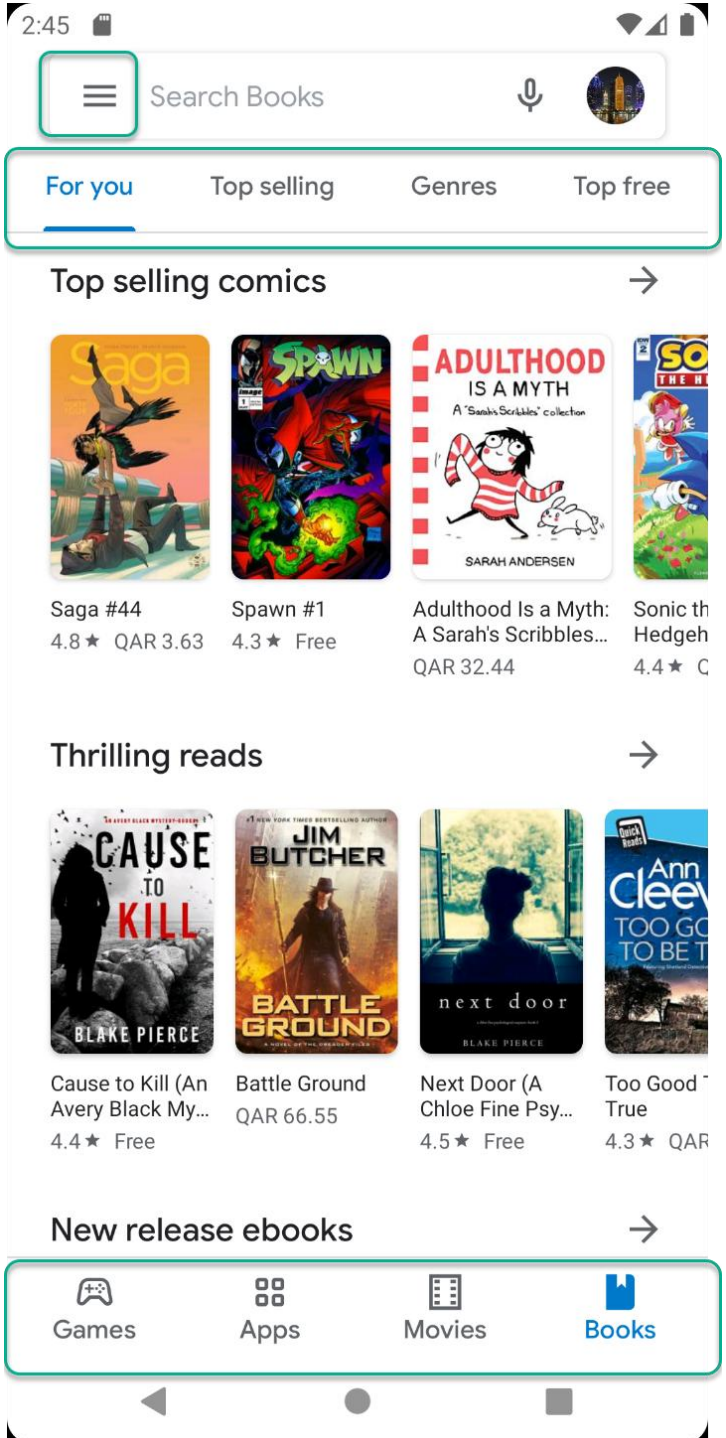
App Bars

BottomNavigationBar

Navigation Rail

Floating Action Button

Navigation Drawer

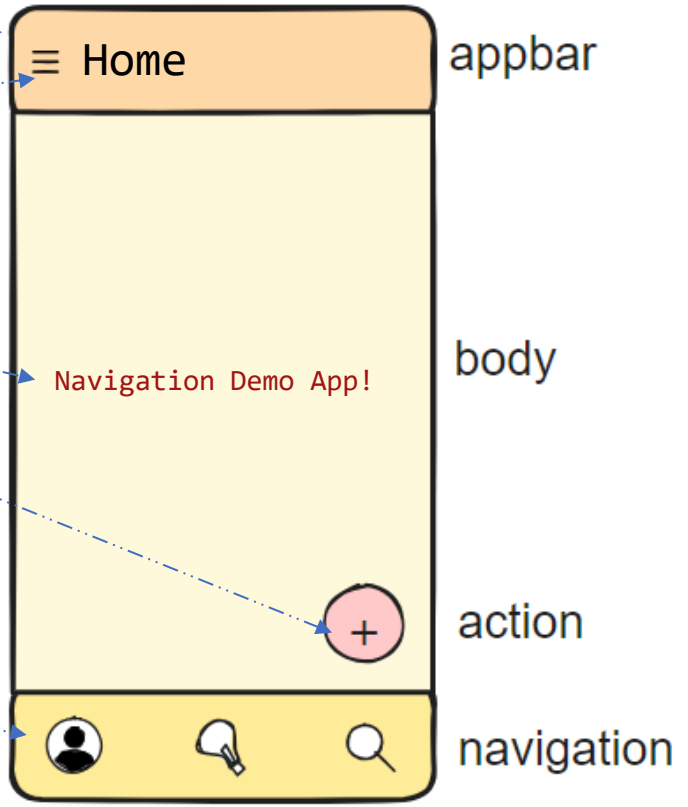


Back

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

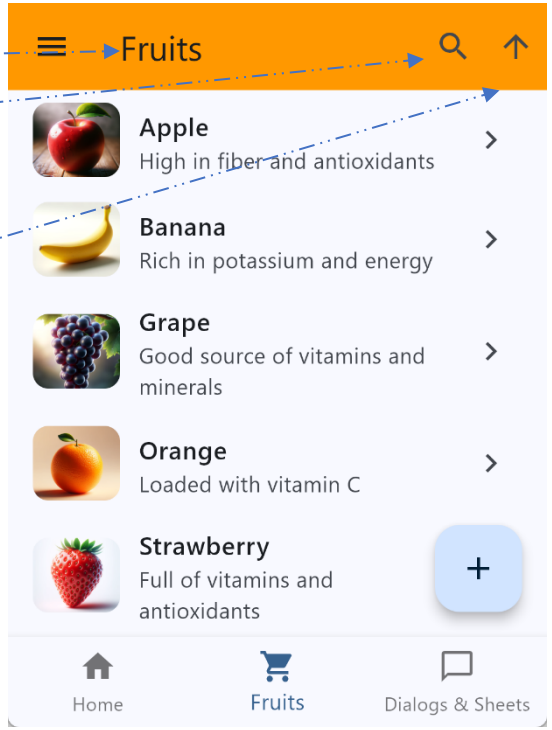
```
Scaffold(
    appBar: AppBar(
      title: const Text('Home'),
    ),
    drawer: const NavDrawer(),
    body: const Center(
      child: Text('Navigation Demo App!'),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: () {
        context.go('add-fruits');
      },
      child: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



AppBar

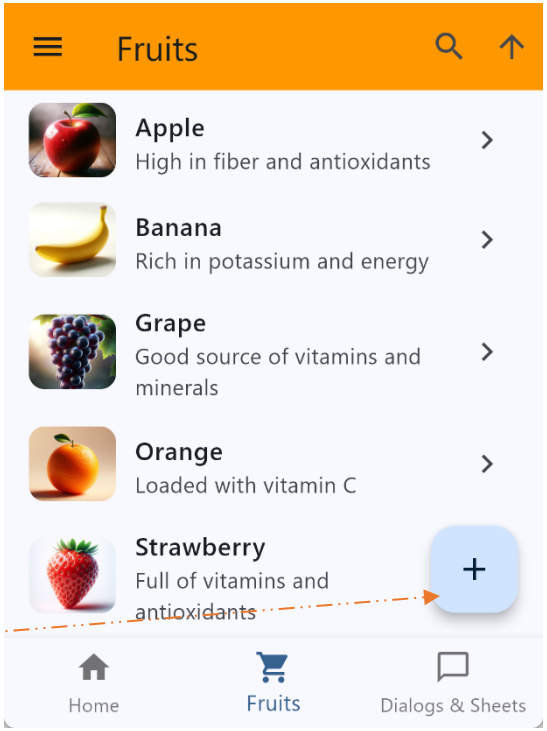
- Info and actions **related to the current screen**
- Typically has Title, Drawer button / Back button, Icon buttons such as Search

```
AppBar(
  title: const Text('Fruits'),
  actions: [
    IconButton(
      onPressed: () {},
      tooltip: 'Search',
      icon: const Icon(Icons.search),
    ),
    IconButton(
      onPressed: () {},
      tooltip: 'Sort',
      icon: const Icon(Icons.sort),
    ),
  ],
)
```



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

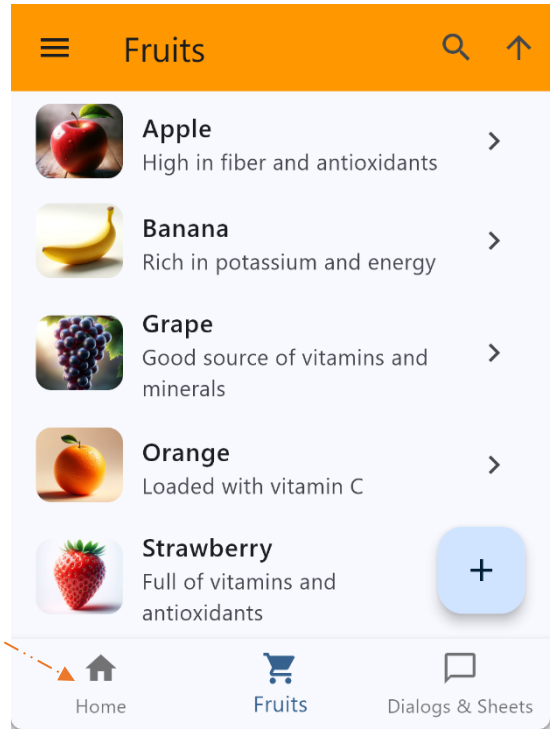



```
FloatingActionButton(
  onPressed: () { ... },
  tooltip: 'Add',
  child: const Icon(Icons.add),
)
```

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

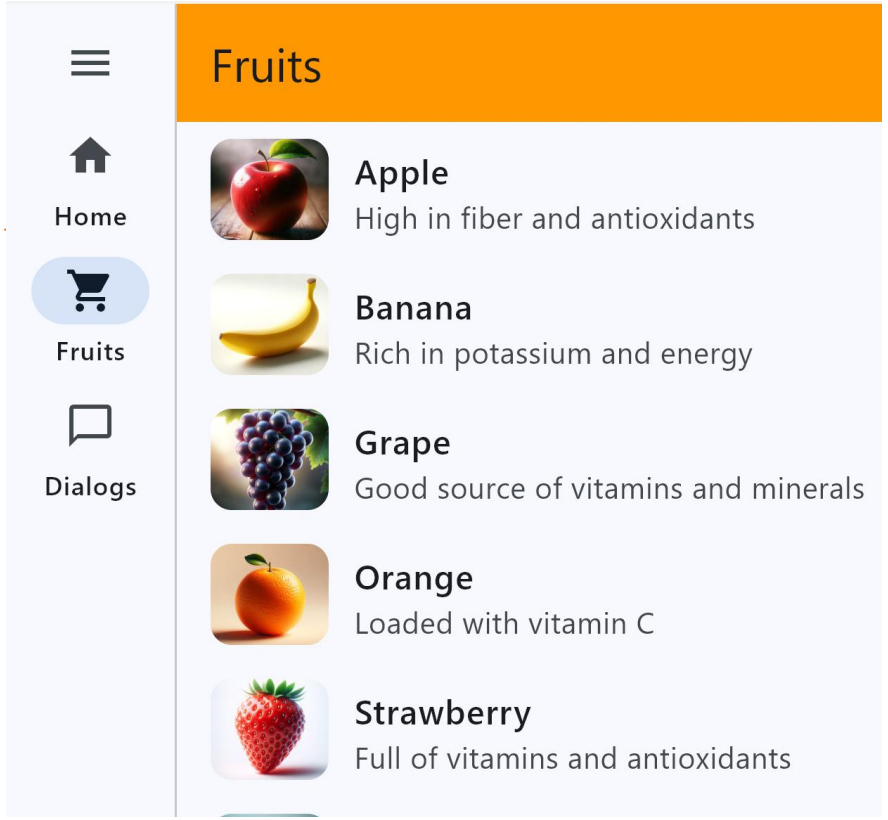

```
BottomNavigationBar(
  currentIndex: 0,
  onTap: (index) { ... },
  items: const [
    BottomNavigationBarItem(
      icon: Icon(Icons.home),
      label: 'Home',
    ),
    BottomNavigationBarItem(
      icon: Icon(Icons.grocery_store),
      label: 'Fruits',
    ),
  ],
),
```



Navigation Rail

- Can contain 3-7 destinations
- Recommended for **medium** or **expanded** screens

```
NavigationRail(
        selectedIndex: 0,
        onDestinationSelected: (index) {},
        labelType: NavigationRailLabelType.all,
        destinations: const [
          NavigationRailDestination(
            icon: Icon(Icons.home),
            label: Text('Home'),
          ),
          NavigationRailDestination(
            icon: Icon(Icons.grocery_store),
            label: Text('Fruits'),
          ),
        ],
)
```

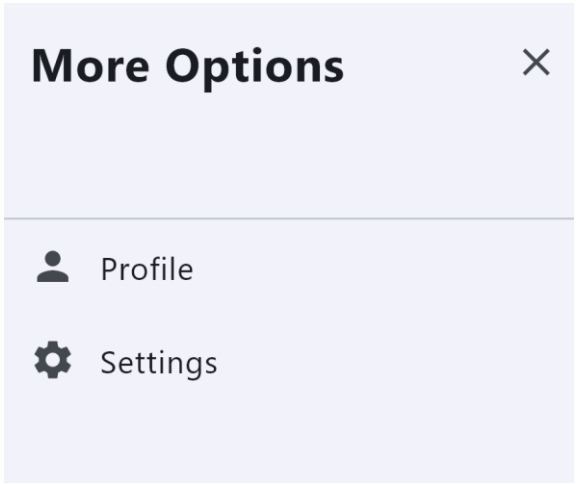


Navigation Drawer

- Navigation Drawer provides access to app **destinations** that cannot fit on the Bottom App 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

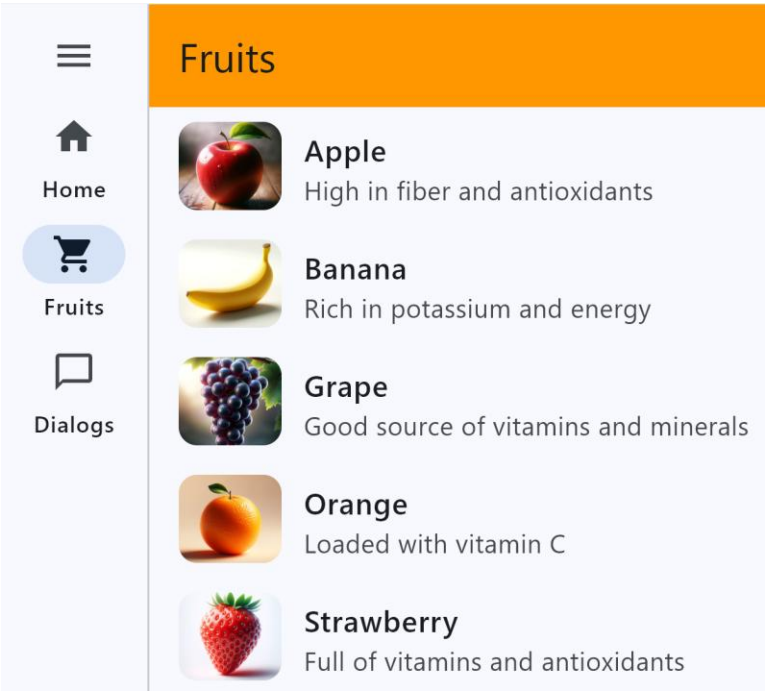
Navigation Drawer - Example

```
Drawer(
   child: ListView(
     padding: EdgeInsets.zero,
     children: [
       const DrawerHeader(
         child: Text('More Options'),
       ),
       ListTile(
         leading: const Icon(Icons.user),
         title: const Text('Profile'),
         onTap: () {},
       ),
       ListTile(
         leading: const Icon(Icons.settings),
         title: const Text('Settings'),
         onTap: () {},
       ),
     ],
   ),
)
```

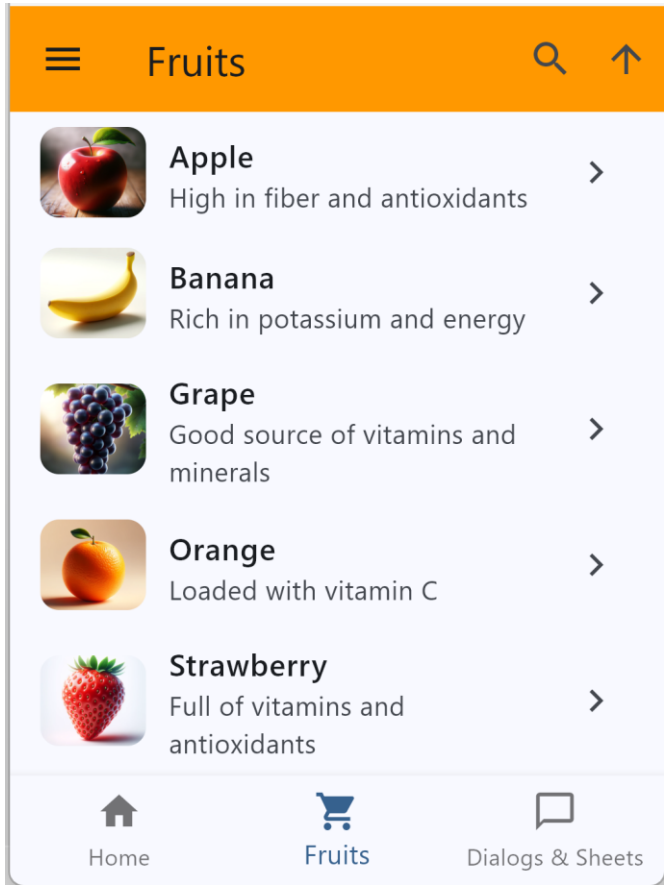


- See more details in the posted example

Adaptive Navigation



Navigation rail on tablet/desktop

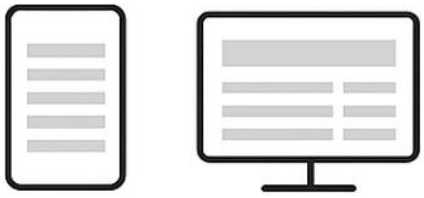


Bottom navigation bar on mobile

Back

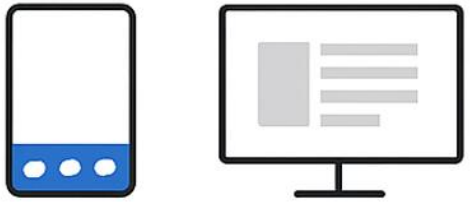
Responsive UI vs. Adaptive Navigation

- **Responsive UI** = Adjusts layout to different screen sizes
  - For an optimal viewing experience across devices (mobile, tablet, desktop, TV)
  - Example: A news app might show a single text column on mobile but multiple columns on larger screens



News app: 1 column on mobile multiple on desktop

- **Adaptive Navigation** = Adjusts navigation structure based on screen size
  - Example: Bottom navigation bar on mobile vs. Navigation rail on tablet/desktop

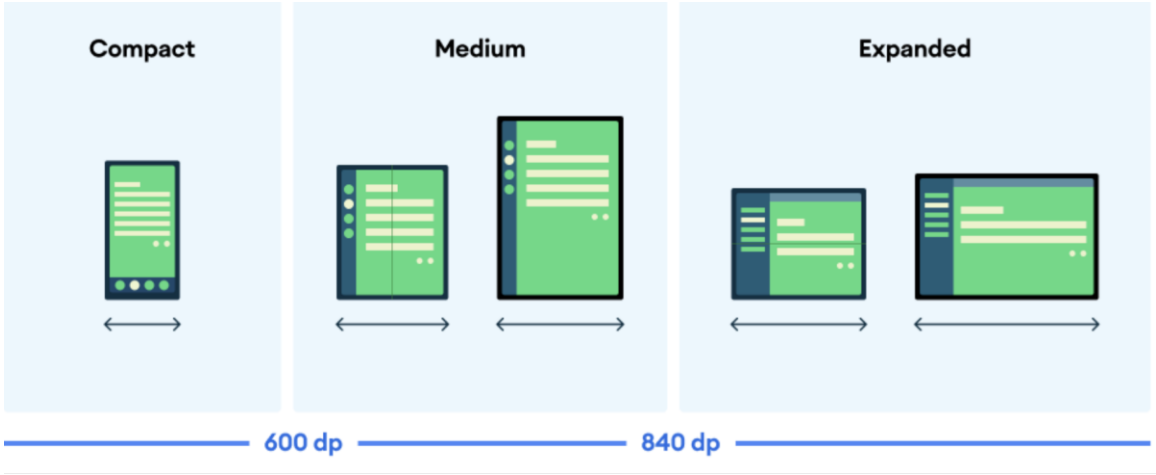


Bottom nav bar on mobile navigation rail on tablet/dektop

24

MediaQuery.of(context).size

- Design for window size classes instead of specific devices
  - Common breakpoints based on screen width

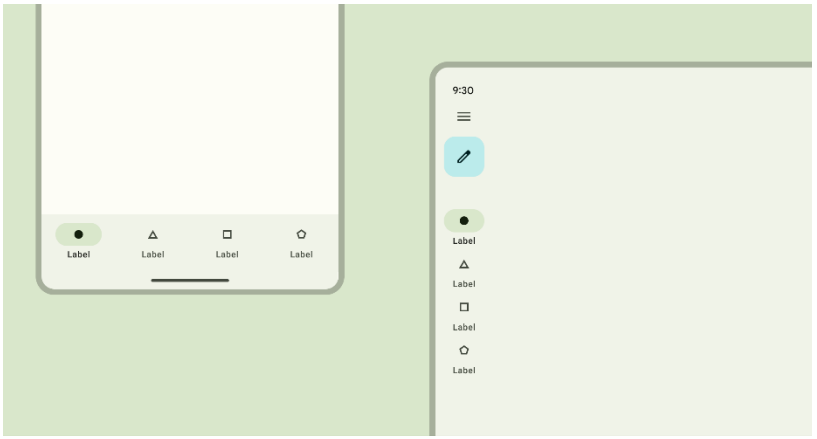


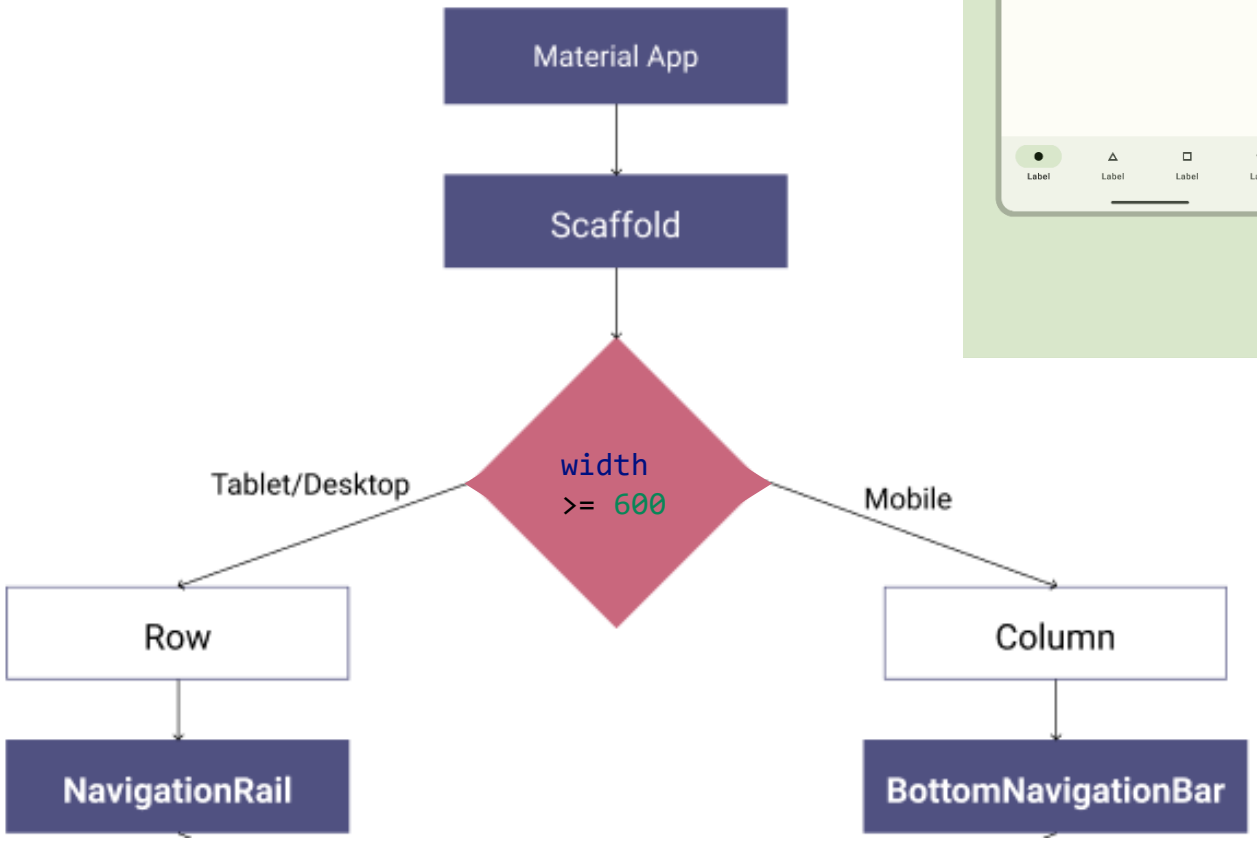
- Use `MediaQuery.of(context).size` to get the window size

```
final isMobile = MediaQuery.of(context).size.width < 600;
final scaffold =
isMobile ? MobileScaffold() : TabletDesktopScaffold();
```

Adaptive Navigation - Example

- In compact screens, use a bottom navigation bar
- Switch to a navigation rail for medium or expanded screens





Adaptive Navigation - Example

```dart
class Home extends StatelessWidget {
  Widget build(BuildContext context) {
    final isMobile = MediaQuery.sizeOf(context).width < 600;
    final content = const Center(child: Text('Home'));
    if (isMobile) {   // Compact: Scaffold + Bottom Navigation Bar
      return Scaffold(
        body: content,
        bottomNavigationBar: BottomNavBar (
          onTap: (_) { ... },
          items: const [
            BottomNavigationBarItem(icon: Icon(Icons.home), label: 'Home'),
            BottomNavigationBarItem(icon: Icon(Icons.apple), label: 'Fruits'),
          ],
        ), );
    }
    // Medium/Expanded: Scaffold + Navigation Rail
    return Scaffold(
      body: Row(
        children: [
          NavigationRail( ...
            onDestinationSelected: (_) { ... },
            destinations: const [
              NavigationRailDestination(icon: Icon(Icons.home), label: Text('Home')),
              NavigationRailDestination(icon: Icon(Icons.apple), label: Text('Fruits')),
            ],
          ),
          const VerticalDivider(width: 1),
          Expanded(child: content),
        ],
      ), );
  }
}
```

See full implementation in **app_router.dart** in the navigation example

Dialogs and Sheets

Overlay Components

- **Overlay components** are temporary UI surfaces that appear above the main content to:
  - ✅ Present **critical information** or choices (non-urgent such as success confirmations use SnackBar instead)
  - ✅ Capture user input without navigation
  - ✅ Provide contextual actions or settings
- Fundamental to modern app User Experience (UX) as they allow users to **stay in context while completing secondary tasks**.
- The key is choosing the right component for the right situation. Three Primary Types:

Component	Visual Position	Best For
Dialog	Center overlay	Urgent decisions, alerts
Bottom Sheet	Bottom edge	Mobile actions, filters
Side Sheet	Right edge	Tablet/Desktop forms, settings

Dialogs - Purpose & Use Cases

Dialog = "Do this task **NOW**"
vs Screen = "Explore freely"

Dialogs interrupt the user flow to:

⚠️ Display critical alerts or errors

`"Error: Payment failed. Retry?" (Alert)`

✅ Request confirmation for destructive actions

`"Delete account? This cannot be undone" (Confirmation)`

🔐 Obtain permissions or consent

`"Allow camera access?" (Permission)`

`"Unsaved changes. Save or discard?" (Decision)`

📝 Capture simple, focused input

`"Select your preferred language" (Simple selection)`

Dialog Types	Purpose
Alert Dialog	Simple decisions, 1-2 actions
Confirmation Dialog	Yes/No, destructive actions
Selection List Dialog	Select one option from a list
Full-Screen Dialog	Complex form

AlertDialog - Key Properties & Events

```
AlertDialog(
      // Optional visual identifier
      icon: const Icon(Icons.delete_outline),
      title: const Text('Delete Item?'), // Required: Clear heading
      // Optional: Supporting text
      content: const Text('Are you sure you want to delete this item?),
      actions: [     // 1-3 action buttons
        TextButton(
          // closes dialog and returns false to caller
          onPressed: () => context.pop(false),
          child: const Text('Cancel'),
        ),
        FilledButton(
          // closes dialog and returns true to caller
          onPressed: () => context.pop(true),
          child: const Text('Delete'),
        ),
      ],
)
```

Showing a Dialog and getting the Returned Value

```
// A dialog may return a simple value
ElevatedButton.icon(
  onPressed: () async {
    final messenger = ScaffoldMessenger.of(context);
    final confirmed = await showDialog<bool>(
      context: context,
      builder: (context) => const AlertDialog( ... ),
    );
    messenger.showSnackBar(
       SnackBar(content: Text("Delete Confirmed: $confirmed")),
    );
}
```

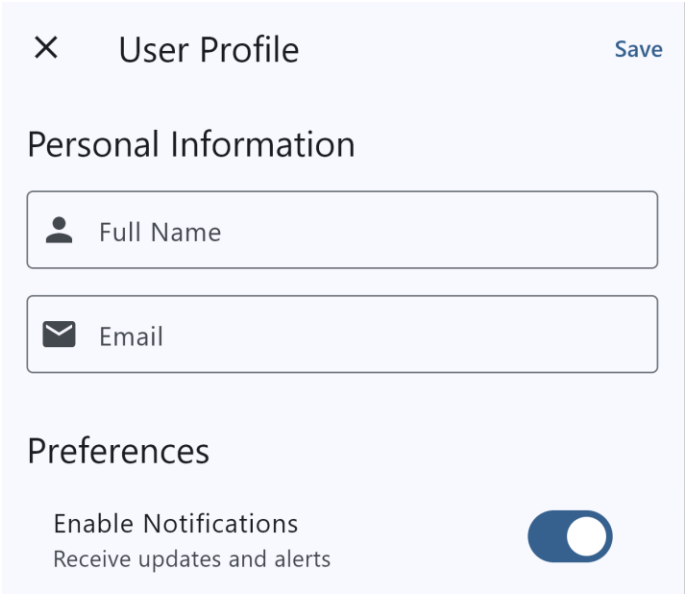
```
// A dialog may return an object
final userProfile = await showDialog<UserProfile>(
  context: context,
  builder: (context) => ProfileEditDialog(),
);
```

Selection List Dialog

```
AlertDialog(
    title: const Text('Select Ringtone'),
    content: SingleChildScrollView(
      child: Column( ...
        children: [
            ListTile(
              leading: const Icon(Icons.music_note),
              title: const Text('Default'),
              onTap: () => context.pop('Default'), // Close + return value
            ),
            ListTile(
              leading: const Icon(Icons.album),
              title: const Text('Classical'),
              onTap: () => context.pop('Classical'),
            ),
          // More options...
        ],
      ),
    ),
)
```

Full-Screen Dialog

- It’s a **modal screen** (blocking and takes up entire screen) for temporary short task that the user must complete or cancel
- The top app bar typically has a **“Close” (✕)** button instead of (←) a back arrow, to indicates a temporary task



🤔 **Use when** you need a **temporary, focused task** related to the current screen, e.g.:

- Editing a profile form (Save or Cancel required)
- Choosing filters or settings (Apply or Discard)
- Composing a message (Send or Discard)

Fullscreen dialog route uses pageBuilder

```
// Fullscreen dialog - uses pageBuilder for custom page transition
GoRoute(
  path: '/profileDialog',
  pageBuilder: (context, state) {
    return MaterialPage(
      // fullscreenDialog: true changes the transition animation
      // - Shows slide-up from bottom (instead of slide from right)
      // - Displays close icon (✕) instead of back arrow (←)
      // - Indicates this is a temporary task, not a destination
      fullscreenDialog: true, // 👈 Key difference from regular navigation
      child: const FullScreenDialog(),
    );
  },
)
```

Bottom Sheets - Purpose & Use Cases

🎯 **Purpose**: Present quick, contextual actions or information without leaving the current screen. Suitable for phones and tablets. On desktop use side sheets.

Key **Use Cases** include:

🔍 Provide filtering or sorting options

- ✓ `Trips filter: Distance, Price, Rating (Standard)`
- ✓ `Products sort: Price, Rating, Newest (Standard)`

📤 Show sharing or export actions

- ✓ `Social media sharing(Standard)`
- ✓ `Photo upload: Camera or Gallery (Modal)`

📝 Capture short-form input (2-5 fields)

- ✓ `Quick note creation with title + body (Modal)`

📋 Display supplementary information

- ✓ `Lectures player queue (Standard)`

Bottom Sheet Variants

Type	Interaction	Use Case
Standard	Non-blocking	Quick information/actions
Modal	Blocking	Focused tasks requiring **user decision**

Bottom Sheets - Example

```dart
// Standard Bottom Sheet
showBottomSheet(
  context: context,
  builder: (context) => MyBottomSheet(),
);

// Modal Bottom Sheet
final result = await showModalBottomSheet(
  context: context,
  isDismissible: true,    // Tap outside to dismiss
  enableDrag: true,       // Swipe down to dismiss
  builder: (context) => MyBottomSheet(),
);
```

Dims the background and prevents interaction

Responsive Design with Sheets

```
// ✓ Good: Adapt to screen size
void _showResponsiveSheet(BuildContext context) {
  final isLargeScreen = MediaQuery.of(context).size.width > 600;
  if (isLargeScreen) {
    // Desktop/tablet: Use side sheet
    _showSideSheet(context);
  } else {
    // Mobile: Use bottom sheet
    showModalBottomSheet(
      context: context,
      builder: (context) => const MySheet(),
    );
}
```

Side Sheets - Purpose & Use Cases

🎯 **Purpose**: Desktop/tablet equivalent of bottom sheets. Present quick, contextual actions or information while interacting with the main content. Key **Use Cases** :

⚙️ Provide detailed settings or preferences

✓ `Dashboard widget settings panel (Standard)`

🔍 Show advanced filtering options

✓ `Product filters: Price, Size, Color (Standard)`

📝 Display contextual forms (5-15 fields)

✓ `Email compose pane in inbox view (Modal)`

✓ `Calendar event editor (Modal)`

📊 Present supplementary data or analytics

🛠️ Offer tool palettes or property editors

✓ `Image editor tool palette (Standard)`

✓ `Document properties and metadata (Standard)`

Side Sheet Variants

Type	Interaction	Use Case
Standard	Non-blocking	Contextual tools, allows interacting with main content
Modal	Blocking	Forms requiring focused input

Side Sheets - Example

```dart
void _showSideSheet(BuildContext context, {bool modal = false}) {
  showGeneralDialog(
    context: context,
    barrierDismissible: true, // Allow tap outside to close
    // 👇 Key difference:
    // modal → dimmed background (requires attention)
    // standard → transparent (non-blocking overlay)
    barrierColor: modal ? Colors.black54 : Colors.transparent,
    pageBuilder: (_, __, ___) {
      return Align(
        alignment: Alignment.centerRight, // Slide in from right edge
        child: Container(
          width: 300,
          color: Colors.white,
          child: const Center(
            child: Text('Side Sheet Content'),
          ),
        ),
      );
    },
  );
}
```

Modal, for forms requiring focused input

Non-modal, user can interact with the main content

Comparison Matrix

Criteria	Dialog	Bottom Sheet	Side Sheet
Purpose	Urgent interruptions, critical decisions	Quick actions, supplementary content	Contextual tools, detailed settings
Modality	Always modal	Modal or Standard	Modal or Standard
Position	Center overlay	Bottom edge	Right edge
Best Screen Size	All sizes	Mobile, Tablet	Tablet, Desktop
Interaction Model	Must respond before continuing	Can dismiss easily (swipe/tap outside)	Can coexist with main content
User Flow	Blocking, requires decision	Supplementary, optional	Parallel, contextual
Use Frequency	Rare (critical moments)	Common (contextual actions)	Moderate (detailed tasks)
Primary Use Cases	Alerts, confirmations, permissions	Filters, shares, quick forms	Filters, settings, complex forms

🌳 Decision Tree

1. **Is it critical/urgent?**
   - ✅ YES → **Dialog**
   - ❌ NO → Continue to #2
2. **What's the screen width?**
   - 📱 < 600dp (Mobile) → Continue to #3
   - 🖥️ ≥ 600dp (Tablet/Desktop) → Continue to #4
3. **Mobile: Does it require a decision?**
   - ✅ YES → **Modal Bottom Sheet**
   - ❌ NO → **Standard Bottom Sheet**
4. **Desktop: Does it require focused input?**
   - ✅ YES → **Modal Side Sheet**
   - ❌ NO → **Standard Side Sheet**

📌 **Special Cases**:

- 🎉 Success message → **SnackBar**
- 📄 Very long form (15+ fields) → **Full Page**
- 🧭 Navigation controls → **App bar** / **FAB** (not overlay)

🎯 Real-World Scenarios

Scenario 1: Product Filters

Context: User browsing product catalog, wants to filter by price, size, color

Decision:

📱 **Mobile**: Modal Bottom Sheet

- **Why**: Touch-friendly, natural swipe
- **Implementation**: `showModalBottomSheet` with FilterChips and RangeSlider

🖥️ **Desktop**: Standard Side Sheet

- **Why**: Horizontal space available, user can see products while filtering
- **Implementation**: `showGeneralDialog` with `barrierColor: Colors.transparent`

Responsive Design:

```
final isDesktop = MediaQuery.of(context).size.width > 600;
if (isDesktop) {
  _showFilterSideSheet(context);
} else {
  _showFilterBottomSheet(context);
}
```

Scenario 2: Destructive Action (Delete Account)

Context: User clicked “Delete Account” in settings
Decision: **Alert Dialog** (Confirmation)
Why:

⚠️ Critical, irreversible action
🛑 Must interrupt user flow: Clear yes/no decision
📱 Works on all screen sizes
✅ **Implementation**:

```
showDialog(context: context,
  barrierDismissible: false,  // Force explicit choice
  builder: (context) => AlertDialog(
   icon: const Icon(Icons.warning, color: Colors.red),
   title: const Text('Delete Account?'),
   content: const Text('This action is permanent. All data will be lost.'),
   actions: [
      TextButton(
        onPressed: () => context.pop(false),
        child: const Text('Cancel'),
      ),
      FilledButton(
        style: FilledButton.styleFrom(
          backgroundColor: Colors.red,
        ),
        onPressed: () => context.pop(true),
        child: const Text('Delete'),
      ),
    ],
  ),
);
```

Scenario 3: Email Compose

Context: User wants to compose email while viewing inbox

Decision:

📱 **Mobile**: Full-Screen Dialog

- **Why**: Needs full attention, multiple fields, not enough space to see the inbox
- **Implementation**: `MaterialPageRoute(fullscreenDialog: true)`

🖥️ **Desktop**: Modal Side Sheet

- **Why**: Can reference inbox while composing, natural email client pattern
- **Implementation**: `showGeneralDialog` with 360dp width

Code Pattern:

```
final isMobile = MediaQuery.of(context).size.width < 600;
if (isMobile) {
   context.push(
    MaterialPageRoute(
      fullscreenDialog: true,
      builder: (context) => const ComposeEmailScreen(),
    ));
} else {
  _showComposeEmailSideSheet(context);
}
```

Scenario 4: Social Media Share

Context: User taps share button on a post

Decision: **Standard Bottom Sheet**

Why:

🎯 Quick action, doesn't require decision

👆 Touch-friendly share icons

🚫 Non-blocking (user can dismiss easily)

Implementation:

```
showModalBottomSheet(context: context,
  showDragHandle: true,
  builder: (context) => Column(
    mainAxisSize: MainAxisSize.min,
    children: [ListTile( leading: const Icon(Icons.copy),
        title: const Text('Copy Link'),
        onTap: () => _shareVia(context, 'copy')),
      ListTile( leading: const Icon(Icons.message),
        title: const Text('Send Message'),
        onTap: () => _shareVia(context, 'message')),
      ListTile( leading: const Icon(Icons.email),
        title: const Text('Email'),
        onTap: () => _shareVia(context, 'email')),
    ],
  ),
);
```

Scenario 5: Dashboard Widget Settings

Context: User clicks settings icon on a dashboard widget

Decision:

📱 **Mobile**: Modal Bottom Sheet

- **Why**: Focused task, doesn't need to see widget while editing

🖥️ **Desktop**: Standard Side Sheet

- **Why**: Can preview changes in real-time while adjusting settings

Implementation (Desktop):

```
showGeneralDialog(context: context,
  barrierColor: Colors.transparent,
  pageBuilder: (context, _, _) {
    return Align(
      alignment: Alignment.centerRight,
      child: WidgetSettingsSideSheet(
          onSettingChanged: (settings) {
            // Real-time preview updates
            setState(() => widgetSettings = settings);
          },
        ));
  },
);
```

Resources

- **go_router** package
- **Flutter Navigation**
  - https://docs.flutter.dev/ui/navigation
- **Dialogs**: m3.material.io/components/dialogs
- **Bottom Sheets**: m3.material.io/components/bottom-sheets
- **Side Sheets**: m3.material.io/components/side-sheets
- **Flutter Material Widgets**: docs.flutter.dev/ui/widgets/material