

CMPS 312



Navigation

Dr. Abdelkarim Erradi
CSE@QU

Navigation

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

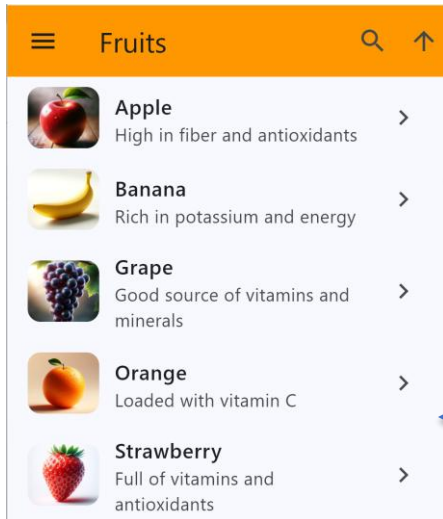
Design effective navigation to
Simplify the user journey

Outline

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

Navigation

Used for navigating between destinations within an app




`.push('/details/3')`

`.pop()`



GoRouter

- First, add [GoRouter](#)  to your `pubspec.yaml`
- Then, **configure** the routes and **integrate** the router with the `MaterialApp`

```
// 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(),
    ),
  ],
);
```

```
// 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

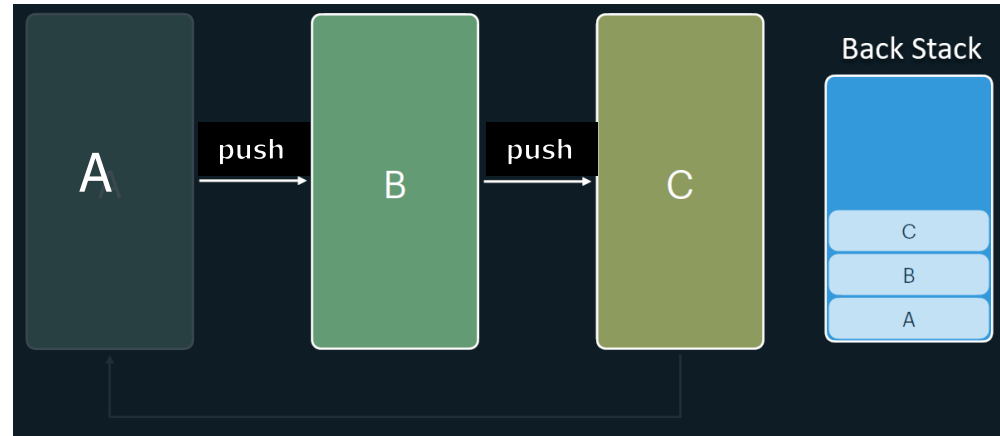
- You can navigate using extension methods on the BuildContext:
 - **context.go()**: Navigates to a new screen, replacing the current route. Good for destinations reached from a BottomNavigationBar or Navigating after successful login
 - **context.push()**: Pushes a new screen onto the top of the navigation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

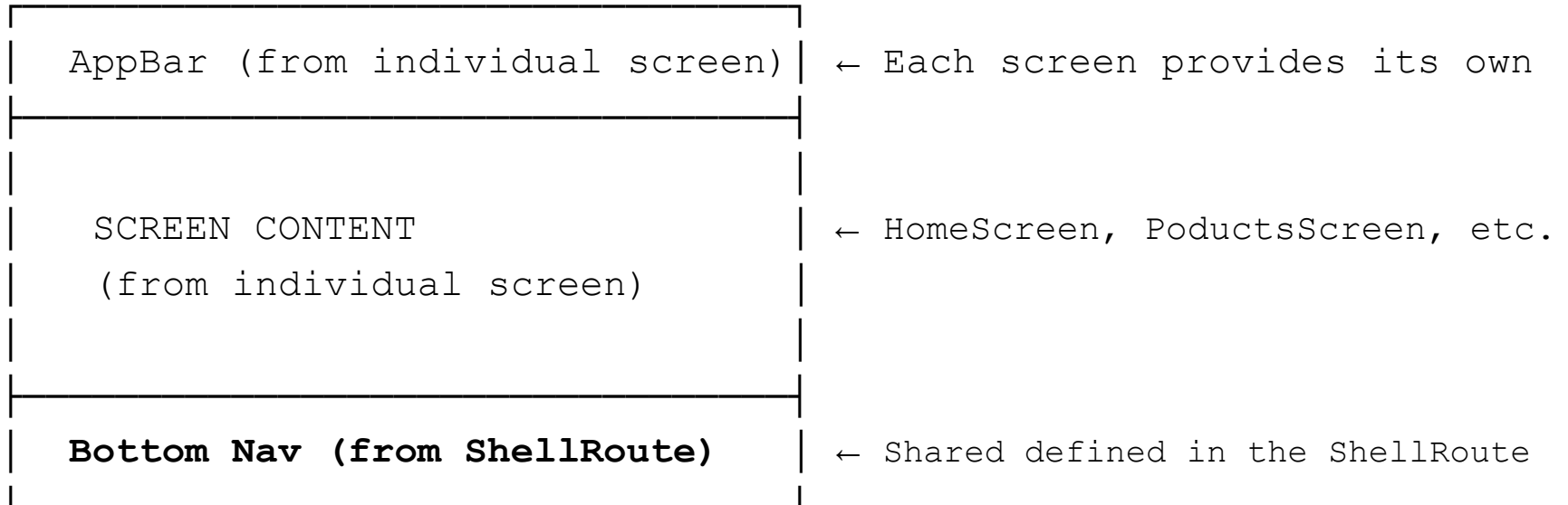
```
context.pop();
```

// 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:

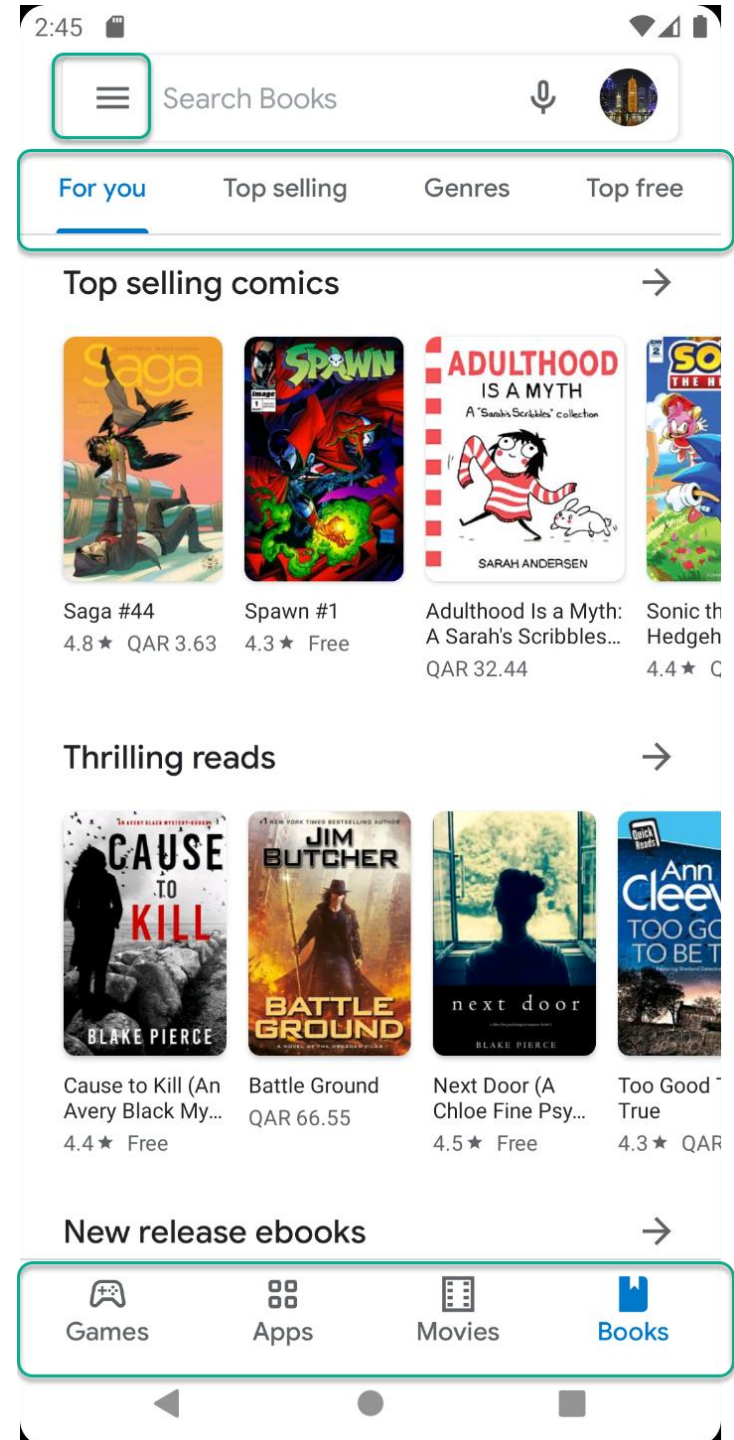
App Bars

BottomNavigationBar

Navigation Rail

Floating Action Button

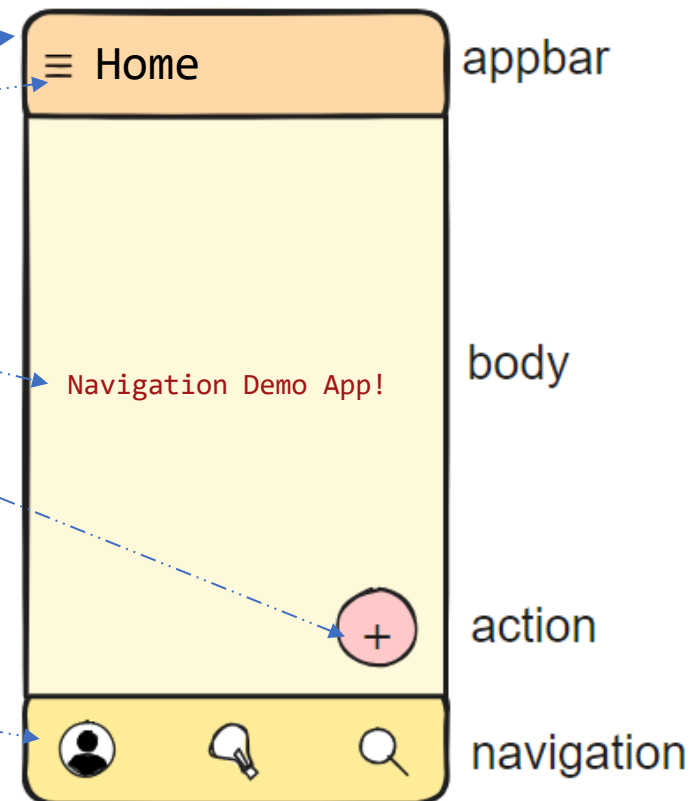
Navigation Drawer



- **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(

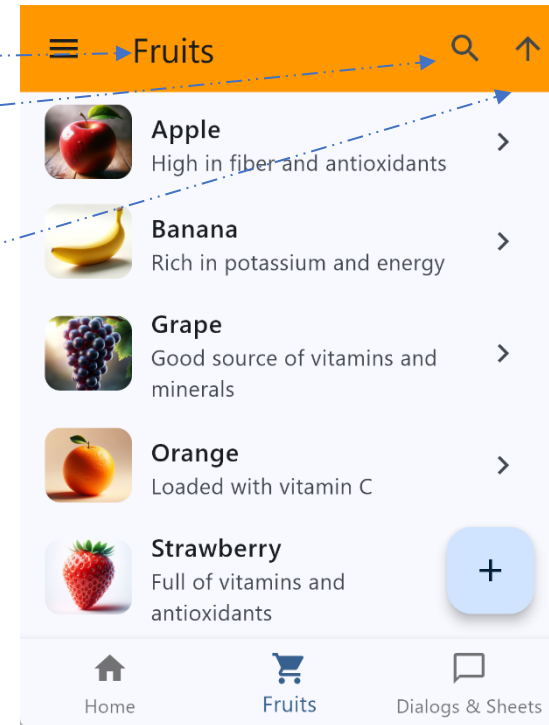
```
  appBar: AppBar(  
    title: const Text('Home'),  
  ),  
  drawer: const NavDrawer(),  
  body: const Center(  
    child: Text('Navigation Demo App!'),  
  ),  
  floatingActionButton: FloatingActionButton(  
    onPressed: () {  
      context.go('add-fruits');  
    },  
    child: const Icon(Icons.local_grocery_store),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

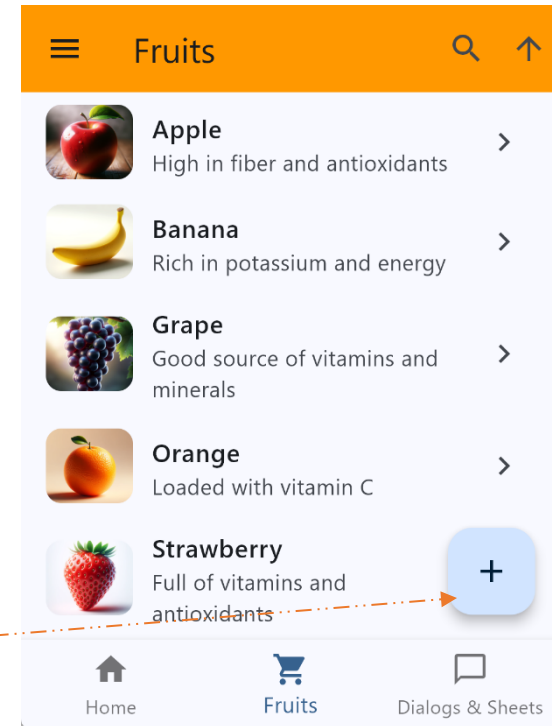
```
AppBar(  
  title: const Text('Fruits'),  
  actions: [  
    IconButton(  
      onPressed: () {},  
      tooltip: 'Search',  
      icon: const Icon(Icons.search),  
    ),  
    IconButton(  
      onPressed: () {},  
      tooltip: 'Sort',  
      icon: const Icon(Icons.sort),  
    ),  
  ],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



Floating Action Button (FAB)

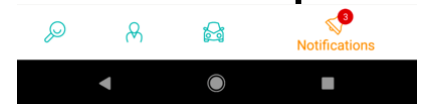
- A FAB performs the primary, or most common, action on a screen, such as drafting a new email
 - It appears in front of all screen content, typically as a circular shape with an icon in its center
 - FAB is typically placed at the bottom right

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

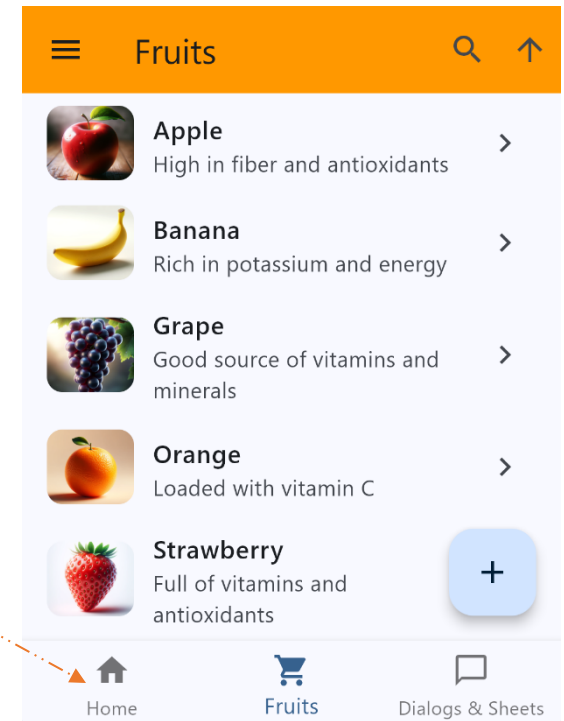


Bottom Navigation Bar

- 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



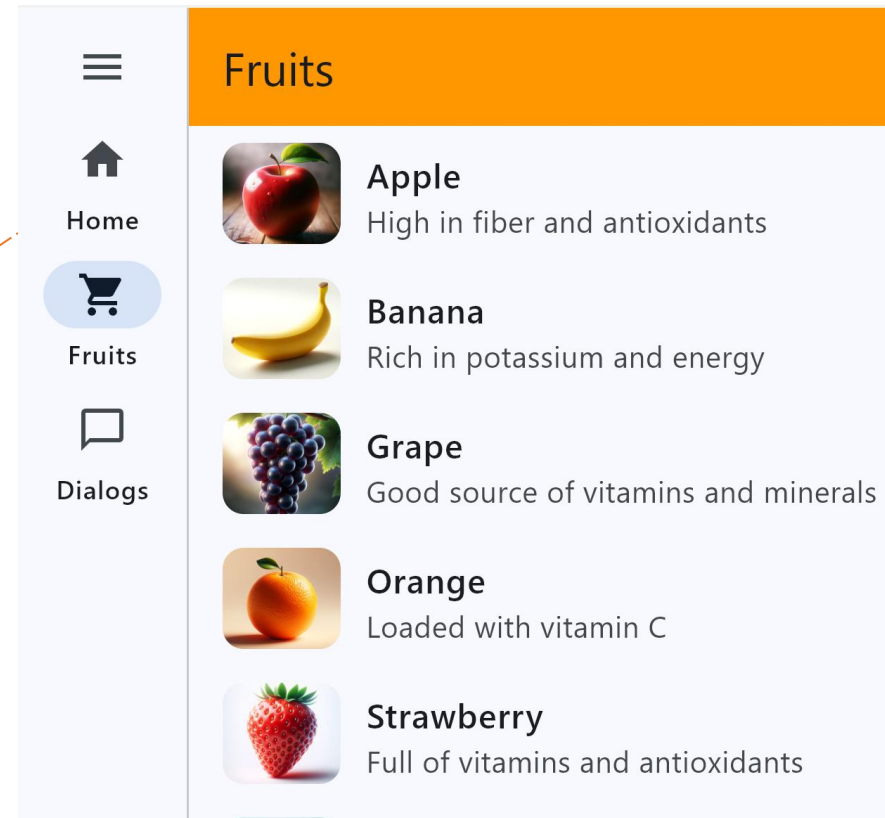
```
BottomNavigationBar(  
  currentIndex: 0,  
  onTap: (index) { ... },  
  items: const [  
    BottomNavigationBarItem(  
      icon: Icon(Icons.home),  
      label: 'Home',  
    ),  
    BottomNavigationBarItem(  
      icon: Icon(Icons.grocery_store),  
      label: 'Fruits',  
    ),  
  ],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



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



القرآن الكريم

MUSHAF TAJWEED

حول التطبيق



تقييم التطبيق



مشاركة التطبيق

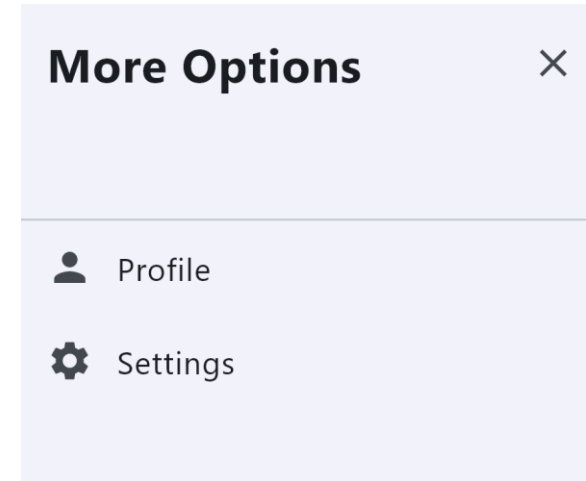


تطبيقات أخرى



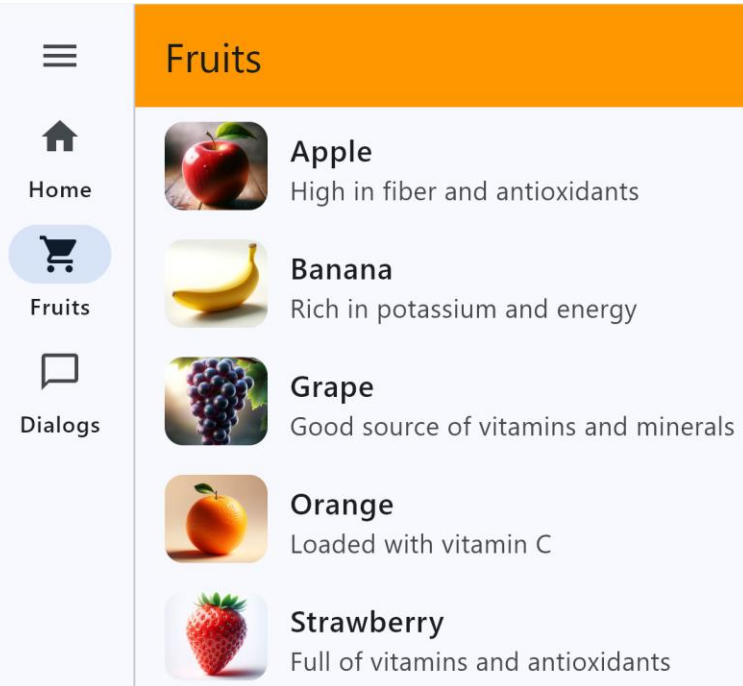
Navigation Drawer - Example

```
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: () {},  
      ),  
    ],  
  ),  
)
```

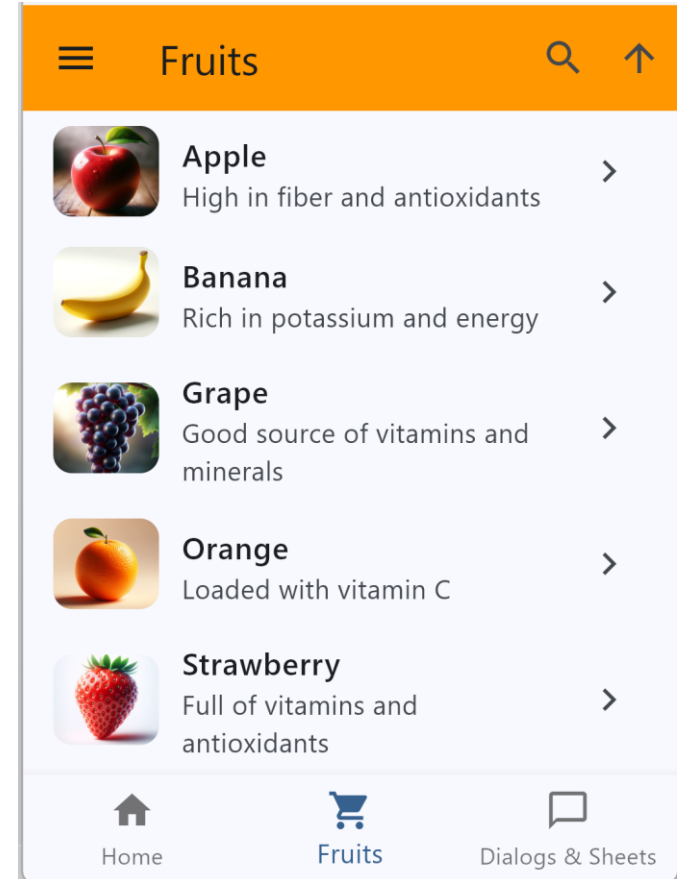


- See more details in the posted example

Adaptive Navigation



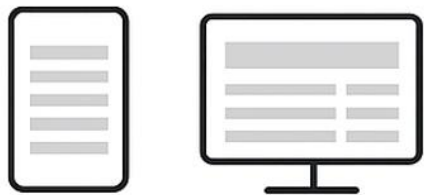
**Navigation rail
on tablet/desktop**



**Bottom navigation bar
on mobile**

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

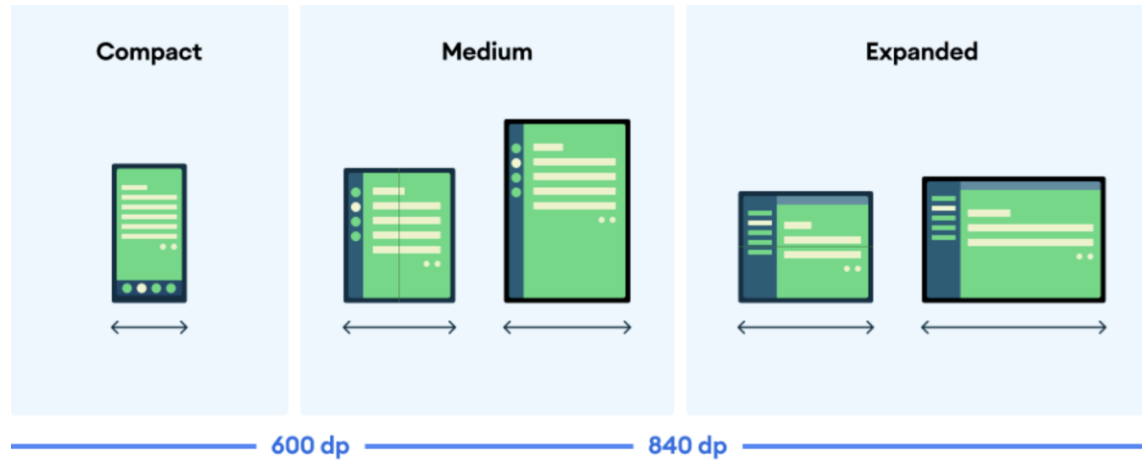
- **Adaptive Navigation** = Adjusts navigation structure based on screen size
 - Example: Bottom navigation bar on mobile vs. Navigation rail on tablet/desktop



Bottom nav bar on mobile
navigation rail on tablet/desktop

MediaQuery.of(context).size

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

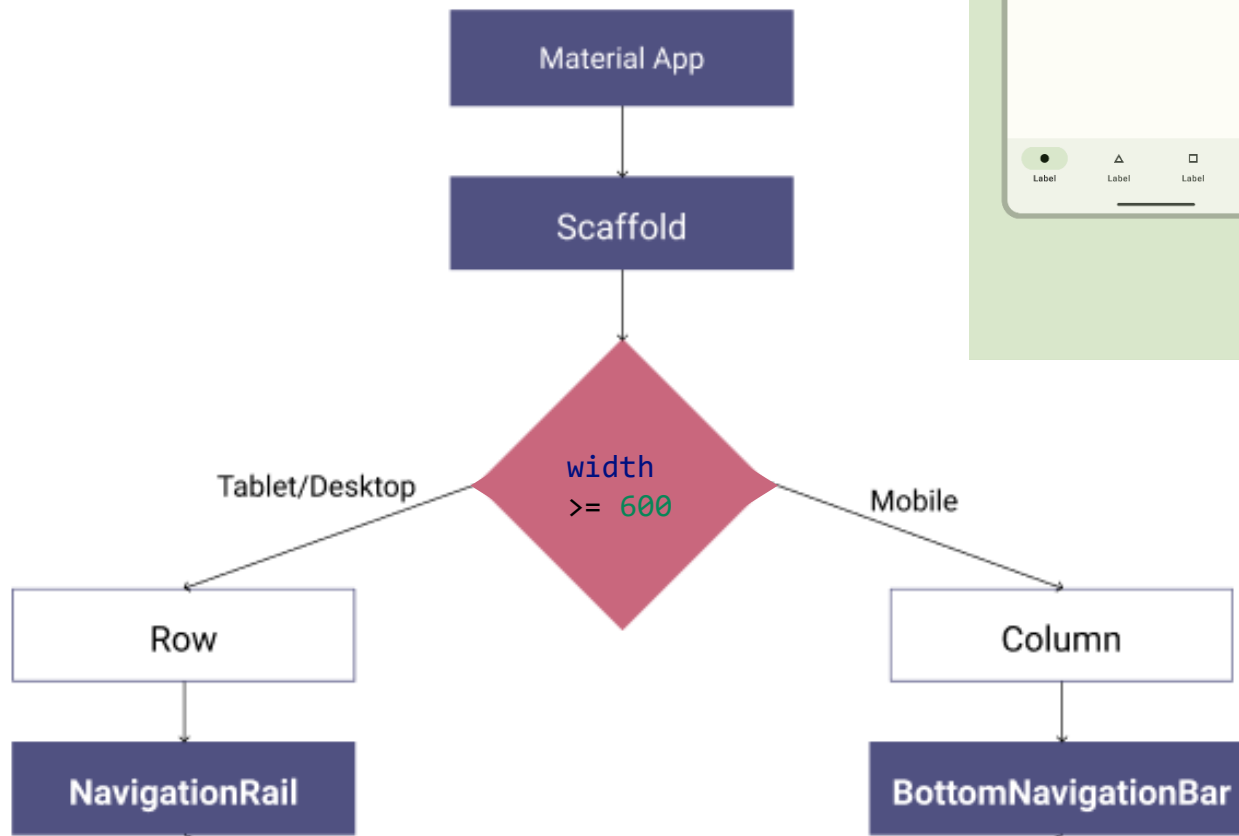


- Use `MediaQuery.of(context).size` to get the window size

```
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



```
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

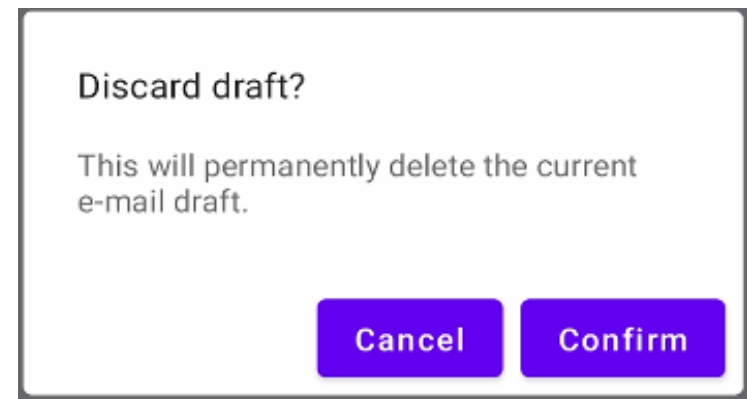


Alert Dialog

- 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

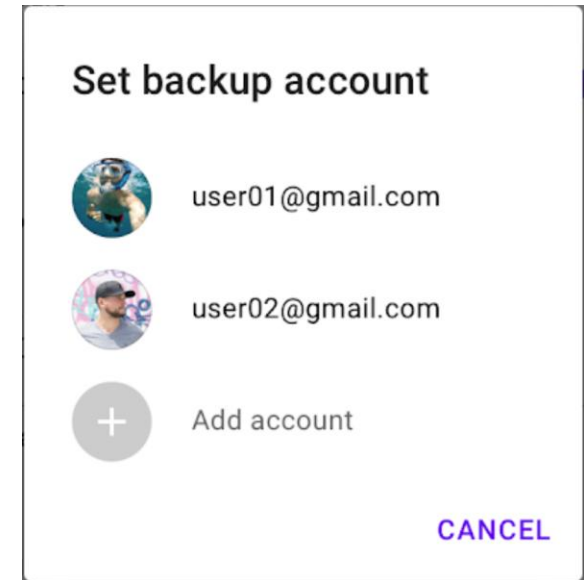
- Commonly used to **confirm high-risk actions** like deleting progress



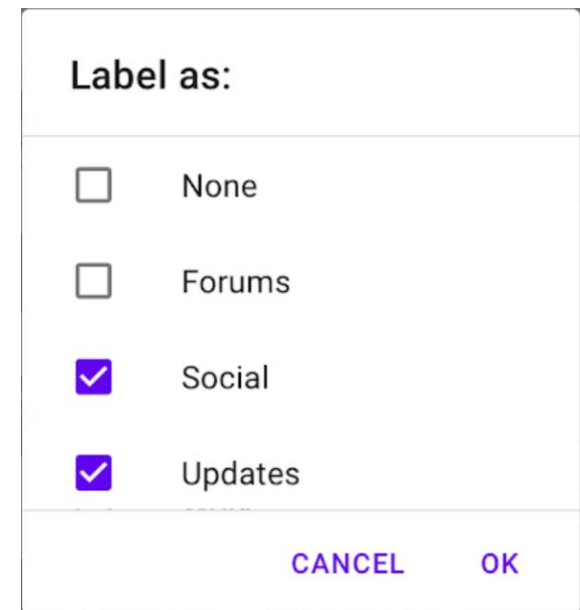
```
AlertDialog(  
    onDismissRequest = {  
        // Dismiss the dialog when the user clicks outside the dialog  
        // or on the back button  
        onDialogOpenChange(false)  
    },  
    title = { Text(text = title) },  
    text = { Text(text = message) },  
    confirmButton = {  
 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

Simple dialog:

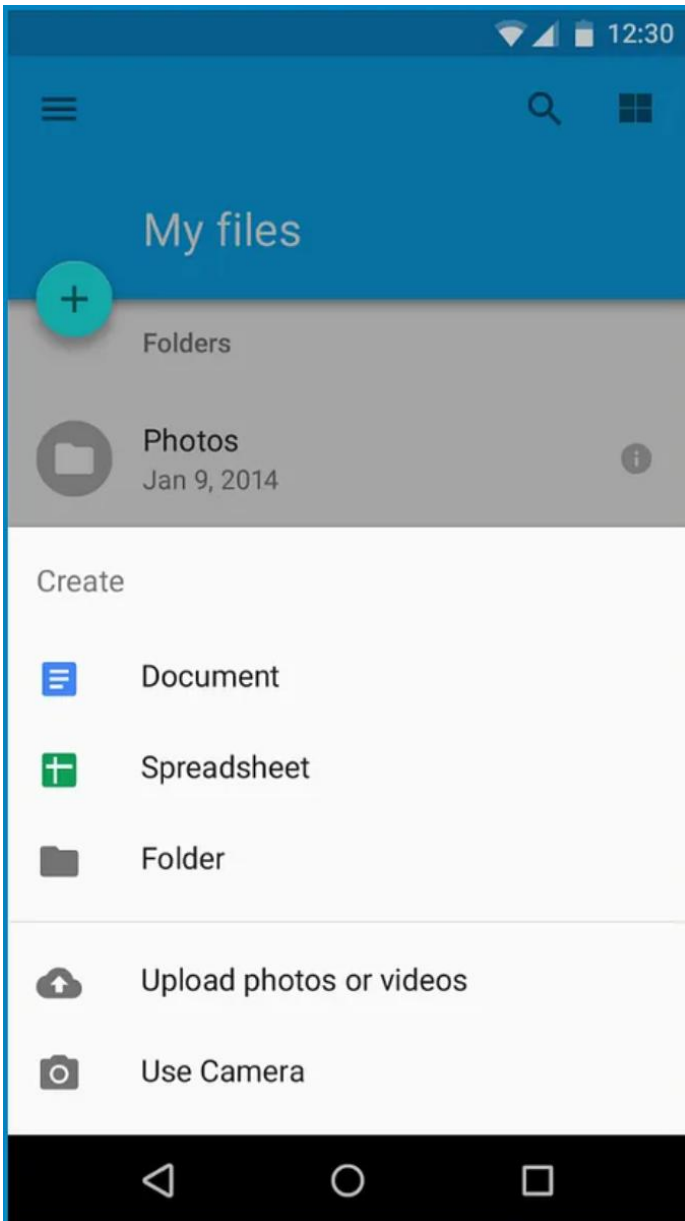
present the user with a list of actions that, when tapped, take immediate effect



Confirmation dialog (multi choice)



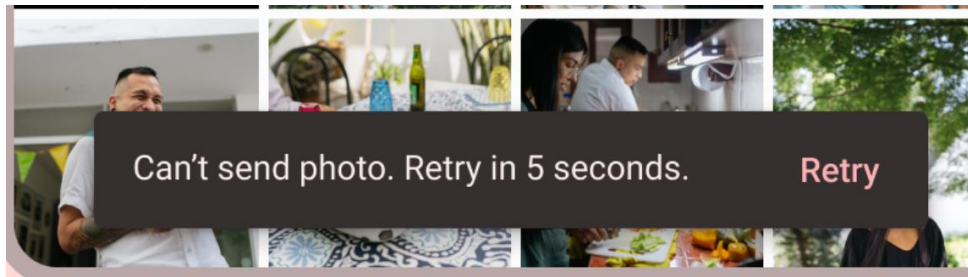
Bottom Sheets



- 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>