🧪 Flutter Lab 3: Navigation in Flutter

🎯 Lab Goals

By completing this lab, the student will be able to:

- Understand classic navigation using **Navigator** (push/pop).
- Use **named routes** and **Router API**.
- Navigate with **GoRouter** (modern Flutter navigation).
- Build apps with **Tabs** and **Drawers**.
- Send data **to** a new screen and receive data **back**.
- Apply best practices to manage routes in medium-sized Flutter apps.

📘 PART 1 — Navigation Fundamentals (Navigator)

1.1 What is Navigation?

Navigation means moving **between screens (`Widgets`)**.
In Flutter, a screen is usually a `Widget` that fills the app window.

Flutter maintains a **navigation stack**:

- `push()` → adds a screen on top
- `pop()` → removes the current screen

Just like a stack of plates.

1.2 Activity A — Push and Pop

Step 1 — Create two screens

`home_screen.dart`

```
import 'package:flutter/material.dart';

class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Home")),
      body: Center(
        child: ElevatedButton(
          child: Text("Go to Details"),
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(builder: (_) => DetailsScreen()),
            );
          },
        ),
      ),
    );
  }
}

class DetailsScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Details")),
      body: Center(
        child: ElevatedButton(
          child: Text("Back"),
          onPressed: () => Navigator.pop(context),
        ),
      ),
    );
  }
}
```

✓ Lab Task

Change the `DetailsScreen` so that pressing “Back" sends a string `"Returned from details"` back to the `HomeScreen`.

1.3 Activity B — Send Data to a Screen

Modify navigation:

```
Navigator.push(
  context,
  MaterialPageRoute(
    builder: (_) => DetailsScreen(message: "Hello from Home!"),
  ),
);
```

`DetailsScreen` receives the data:

```
class DetailsScreen extends StatelessWidget {
  final String message;

  DetailsScreen({required this.message});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Details")),
      body: Center(
        child: Text(message),
      ),
    );
  }
}
```

1.4 Activity C — Receive Data Back From a Screen

Step 1 — Push a route and wait for result

```
final result = await Navigator.push(
  context,
  MaterialPageRoute(builder: (_) => InputScreen()),
);

print("Returned value = $result");
```

Step 2 — The second screen sends data back

```
Navigator.pop(context, "User typed: $text");
```

✔ Practice Task

Create a form screen where the user enters their name, and the home screen displays it after returning.

📘 PART 2 — Named Routes

Why Named Routes?

They make navigation cleaner in larger apps, but they can’t be used with deep links so the best practice is to use a Route package like go_router.

Step 1 — Register routes

```
MaterialApp(
  initialRoute: '/',
  routes: {
    '/': (_) => HomeScreen(),
    '/profile': (_) => ProfileScreen(),
    '/settings': (_) => SettingsScreen(),
  },
);
```

Step 2 — Navigate to a name

```
Navigator.pushNamed(context, "/profile");
```

Task

Navigate to `/settings` when pressing a button.

📘 PART 3 — Router API (GoRouter)

GoRouter is the modern Flutter navigation package.

3.1 Setup

```
pubspec.yaml

dependencies:
  go_router: ^14.0.0
```

3.2 Create Router

```
final _router = GoRouter(
  routes: [
    GoRoute(
      path: '/',
      builder: (_, __) => HomeScreen(),
    ),
    GoRoute(
      path: '/details/:id',
      builder: (_, state) {
        final id = state.pathParameters['id']!;
        return DetailsScreen(id: id);
      },
    ),
  ],
);
```

3.3 Use the Router

```
MaterialApp.router(
  routerConfig: _router,
);
```

3.4 Navigate with GoRouter

Navigate with path data

```
context.go('/details/42');
```

OR push

```
context.push('/details/42');
```

Receive Data

```
class DetailsScreen extends StatelessWidget {
  final String id;

  DetailsScreen({required this.id});
}
```

✔ Task

Modify the path to `/product/:category/:id`.

📘 PART 4 — Tabs Navigation

Tabs use `TabBar` + `TabBarView`.

Example

```
class TabsScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return DefaultTabController(
      length: 3,
      child: Scaffold(
        appBar: AppBar(
          title: Text("Tabs"),
          bottom: TabBar(
            tabs: [
              Tab(icon: Icon(Icons.home), text: "Home"),
              Tab(icon: Icon(Icons.person), text: "Profile"),
              Tab(icon: Icon(Icons.settings), text: "Settings"),
            ],
          ),
        ),
        body: TabBarView(
          children: [
            Center(child: Text("Home Tab")),
            Center(child: Text("Profile Tab")),
            Center(child: Text("Settings Tab")),
          ],
        ),
      ),
    );
  }
}
```

✔ Lab Tasks

- Replace tab content with real screens.
- Add a FAB that changes text only inside the current tab.

📘 PART 5 — Drawer Navigation

A Drawer is the side panel that slides from the left.

Example

```
Scaffold(
  appBar: AppBar(title: Text("Drawer Demo")),
  drawer: Drawer(
    child: ListView(
      children: [
        DrawerHeader(
          child: Text("Menu"),
        ),
        ListTile(
          title: Text("Profile"),
          onTap: () {
            Navigator.pop(context);  // close drawer
            Navigator.pushNamed(context, "/profile");
          },
        ),
      ],
    ),
  ),
  body: Center(child: Text("Home")),
);
```

✔ Task

Add a "Settings" page in the drawer.

📘 PART 6 — Mini Project

🏗 Build a Multi-Navigation App

Your app must include:

✔ 1. Home page

- Buttons that navigate using **Navigator.push**
- A button that navigates using **GoRouter**

✔ 2. Drawer

- Navigate to Profile
- Navigate to Settings

✔ 3. Tabs

- Home
- Products
- Favorites

✔ 4. Communication

- “Add Product” screen
- Return the product name back to the Products tab.

🧠 What the Student Should Understand After This Lab

By the end, the student must be able to:

- Explain **Navigator vs Router**.
- Build screens with **push/pop**.
- Use **GoRoutes** with path parameters.
- Build apps with **Tabs** and **Drawers**.
- Send and receive data when navigating.
- Plan navigation architecture in a real project.