# Types of Navigation in Flutter

## Types of Navigation in Flutter

### 1. Basic Navigation with `Navigator`

Flutter uses a **stack-based navigation system** where you can "push" a new screen onto the stack, and when you are done with a screen, you can "pop" it off. Here's the basic way to navigate using the `Navigator` widget.

#### Push and Pop

To navigate between screens, you typically use `Navigator.push()` to push a new screen onto the stack and `Navigator.pop()` to remove a screen from the stack.

#### Example:

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: FirstScreen(),
    );
  }
}

class FirstScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("First Screen")),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            // Navigate to the second screen
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => SecondScreen()),
            );
          },
          child: Text("Go to Second Screen"),
        ),
      ),
    );
  }
}

class SecondScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Second Screen")),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            // Pop the current screen off the stack (go back)
            Navigator.pop(context);
          },
          child: Text("Back to First Screen"),
        ),
      ),
    );
  }
}
```

### Explanation:

- **Navigator.push()**: Adds a new route (screen) onto the navigation stack.
- **Navigator.pop()**: Pops the top screen off the stack, returning to the previous screen.

---

## 2. Named Routes

Named routes are a convenient way to handle navigation, especially as the app grows larger and you have many different screens. Instead of passing the widget directly, you use route names and define these in a central place.

### Example:

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      initialRoute: '/',
      routes: {
        '/': (context) => HomeScreen(),
        '/second': (context) => SecondScreen(),
      },
    );
  }
}

class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Home Screen")),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            // Navigate to the second screen using the named route
            Navigator.pushNamed(context, '/second');
          },
          child: Text("Go to Second Screen"),
        ),
      ),
    );
  }
}

class SecondScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Second Screen")),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            // Go back to the home screen
            Navigator.pop(context);
          },
          child: Text("Back to Home Screen"),
        ),
      ),
    );
  }
}
```

## Explanation:

**Navigator.pushNamed()**: Navigates to a named route.

- **MaterialApp** widget has a `routes` parameter to define a set of named routes.
- **Navigator.pop()**: Navigates back (pops the current screen off the stack).

---

## 3. Passing Data Between Screens

You can also pass data between screens when navigating.

### Example (Passing Data):

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: HomeScreen(),
    );
  }
}

class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Home Screen")),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            // Pass data to the second screen
            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) => SecondScreen(data: "Hello from Home!"),
              ),
            );
          },
          child: Text("Go to Second Screen"),
        ),
      ),
    );
  }
}
```

```
class SecondScreen extends StatelessWidget {
  final String data;
  SecondScreen({required this.data});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Second Screen")),
      body: Center(
        child: Text("Data from Home: $data"),
      ),
    );
  }
}
```

### Explanation:

- We pass data using the constructor of `SecondScreen`.
- **Navigator.push()** can take a custom `MaterialPageRoute` that includes the data you want to pass.

---

## 4. Navigating Back to a Specific Screen (Using `popUntil`)

Sometimes you need to navigate back to a specific screen, and not just the previous one. You can use `Navigator.popUntil()` to go back to a specific screen based on a condition.

### Example:

```
dart
CopyEdit
Navigator.popUntil(context, ModalRoute.withName('/home'));
```

This will pop screens off the stack until the one with the name '/home' is at the top of the stack.

---

## 5. Using `pushReplacement` and `popAndPushNamed`

Sometimes, you want to replace the current screen with a new screen, rather than just adding it to the stack. You can use `pushReplacement()` or `popAndPushNamed()`.

### Example with `pushReplacement`:

```
dart
CopyEdit
Navigator.pushReplacement(
  context,
  MaterialPageRoute(builder: (context) => NewScreen()),
);
```

This replaces the current screen with the new one.

### Example with `popAndPushNamed`:

```
dart
CopyEdit
Navigator.popAndPushNamed(context, '/newScreen');
```

This pops the current screen off the stack and then pushes a new named route.

## 6. Bottom Navigation and Tab Navigation

In many apps, there are multiple screens you can switch between using a bottom navigation bar or tabs.

### Example with Bottom Navigation Bar:

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: BottomNavScreen(),
    );
  }
}

class BottomNavScreen extends StatefulWidget {
  @override
  _BottomNavScreenState createState() => _BottomNavScreenState();
}

class _BottomNavScreenState extends State<BottomNavScreen> {
  int _selectedIndex = 0;
  final List<Widget> _screens = [
    HomeScreen(),
    SearchScreen(),
    ProfileScreen(),
  ];

  void _onItemTapped(int index) {
    setState(() {
      _selectedIndex = index;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: _screens[_selectedIndex],
      bottomNavigationBar: BottomNavigationBar(
        currentIndex: _selectedIndex,
        onTap: _onItemTapped,
        items: const <BottomNavigationBarItem>[
          BottomNavigationBarItem(
            icon: Icon(Icons.home),
            label: 'Home',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.search),
            label: 'Search',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.account_circle),
            label: 'Profile',
          ),
        ],
      ),
    );
  }
}

class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(child: Text("Home Screen"));
  }
}

class SearchScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(child: Text("Search Screen"));
  }
}

class ProfileScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(child: Text("Profile Screen"));
  }
}
```

## Explanation:

- This creates a **bottom navigation bar** that allows users to switch between different screens (home, search, and profile).

- The **_onItemTapped** method updates the screen based on the selected index.

---

## Conclusion

Flutter provides multiple ways to handle navigation. Whether you use **push/pop**, **named routes**, **bottom navigation**, or **tabs**, it's easy to manage screen transitions and pass data between screens. You can also replace screens, pop multiple routes, or implement advanced navigation strategies like **deep linking** and **nested navigation**.

Let me know if you need further examples or explanations about specific navigation techniques! 😄