Flutter Navigation: Routes, Screens & Passing Data

Subtitle: Learn how to move between screens and build multi-page Flutter apps with proper navigation.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Single-screen apps are useless in real-world scenarios. Every practical app requires multiple screens and smooth navigation between them. This guide teaches you how to **navigate between pages, manage routes, and pass data** in Flutter.

Step 1: Understanding Navigation Basics

Flutter uses a **Navigator stack**:

- New screen → pushed onto stack
- Back action → pops screen from stack

Think of it like:
👉 Screen A → Screen B → Screen C

Step 2: Navigating Between Screens

Basic Navigation (Push)

```
Navigator.push(
  context,
  MaterialPageRoute(builder: (context) => SecondScreen()),
);
```

Go Back (Pop)

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

Exercise: Create two screens and navigate from Screen A → Screen B → back.

Step 3: Named Routes

Instead of writing routes everywhere, define them centrally.

Setup in MaterialApp:

```
MaterialApp(
  initialRoute: '/',
  routes: {
    '/': (context) => HomeScreen(),
    '/second': (context) => SecondScreen(),
  },
)
```

Navigate using name:

```
Navigator.pushNamed(context, '/second');
```

Exercise: Add a third screen using named routes.

Step 4: Passing Data Between Screens

Pass Data (Push)

```
Navigator.push(
  context,
  MaterialPageRoute(
    builder: (context) => SecondScreen(data: "Hello"),
  ),
);
```

Receive Data

```
class SecondScreen extends StatelessWidget {
  final String data;

  SecondScreen({required this.data});

  @override
  Widget build(BuildContext context) {
    return Text(data);
  }
}
```

Exercise: Pass a username from one screen and display it on another.

Step 5: Returning Data Back

You can send data back when popping a screen.

```
Navigator.pop(context, "Returned Data");
```

Receive it:

```
final result = await Navigator.push(...);
print(result);
```

Exercise: Send a selected item back to the previous screen.

Step 6: Best Practices

- Use **named routes** for larger apps
- Keep navigation logic **simple and consistent**
- Avoid deeply nested navigation flows
- Use **clear screen naming conventions**
- Manage navigation with state tools for complex apps

Step 7: Mini Project

Build a **Multi-Screen App**:

Features:

- Home Screen (list of items)
- Details Screen (shows selected item)
- Add Item Screen
- Pass data between screens
- Navigate back with updated data

Key Takeaways

- Flutter uses a **stack-based navigation system**
- Use `Navigator.push()` and `Navigator.pop()`
- Named routes simplify navigation in larger apps
- Data can be passed forward and backward
- Navigation is essential for real-world apps

If you don’t understand navigation properly, your apps will quickly become messy and hard to scale. This is a core skill, not optional.

Visit **haas.dev** for more Flutter guides, tutorials, and complete project resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app
