Mobile Programming

Flutter Navigation & Routing

Lecturer: Ali Najimi
Faculty of Computer Engineering
Sharif University of Technology
Fall 2027

Navigator | GoRouter | Deep Links | Guards



Table of Contents

1. Navigation Fundamentals
2. Imperative Navigation (Navigator 1.0)
3. Declarative Navigation (Navigator 2.0)
4. GoRouter — The Modern Standard
5. Route Parameters & Query Params
6. Nested Navigation & Shell Routes
7. Navigation Guards & Auth
8. Deep Links (Android & iOS)
9. Adaptive Navigation Patterns
10. Mini Lab: Multi-Flow App
11. Summary & Key Takeaways

Navigation Fundamentals

Navigation is how users move between screens in your app.

Core Concepts

- **Route:** A screen/page in the app.
- **Stack:** Routes are managed as a LIFO stack.
- **Push:** Add a new route on top.
- **Pop:** Remove the top route and go back.

Two Paradigms

- **Imperative:** You tell the navigator *what to do* (`push`, `pop`).
- **Declarative:** You describe *what the route stack looks like* based on state.

Flutter supports both — but declarative is the future.

Imperative Navigation (Navigator 1.0)

The original Flutter navigation API. Simple but limited.

```dart
// Push a new screen
Navigator.push(
  context,
  MaterialPageRoute(builder: (_) => const DetailScreen()),
);

// Pop back
Navigator.pop(context);

// Push with a result
final result = await Navigator.push(
  context,
  MaterialPageRoute(builder: (_) => const PickerScreen()),
);
// Handle result
if (result != null) { /* use result */ }
```

Problem: Hard to handle deep links, browser URL bar, and complex navigation flows.

Named Routes

A step toward declarative — define routes as a map.

```
MaterialApp(
  routes: {
    '/': (context) => const HomeScreen(),
    '/details': (context) => const DetailScreen(),
    '/settings': (context) => const SettingsScreen(),
  },
);

// Navigate by name
Navigator.pushNamed(context, '/details');

// With arguments
Navigator.pushNamed(context, '/details', arguments: {'id': 42});
```

Limitation: Still imperative. No type safety. Hard to manage nested flows or guards.

Navigator 2.0 (Declarative)

Flutter's new API — you describe the route stack based on state.

```
class MyRouterDelegate extends RouterDelegate<RoutePath>
    with ChangeNotifier, PopNavigatorRouterDelegateMixin {
  @override
  Widget build(BuildContext context) {
    return Navigator(
      pages: [
        MaterialPage(child: HomeScreen()),
        if (selectedItem != null)
          MaterialPage(child: DetailScreen(item: selectedItem!)),
      ],
      onPopPage: (route, result) {
        selectedItem = null;
        notifyListeners();
        return route.didPop(result);
      },
    );
  }
}
```


Problem: Very verbose. This is why GoRouter exists.

GoRouter — The Modern Standard

GoRouter is the **official Flutter package** for declarative routing. It simplifies Navigator 2.0.

Why GoRouter?

- **Declarative** route definitions.
- **Deep linking** built-in.
- **Redirects & guards** for auth.
- **Nested routes** for complex layouts.
- **Type-safe** route parameters.

```
dependencies:
  go_router: ^14.0.0
```

GoRouter — Basic Setup

```
final router = GoRouter(
  routes: [
    GoRoute(
      path: '/',
      builder: (context, state) => const HomeScreen(),
    ),
    GoRoute(
      path: '/details',
      builder: (context, state) => const DetailScreen(),
    ),
    GoRoute(
      path: '/profile/:userId',
      builder: (context, state) {
        final userId = state.pathParameters['userId']!;
        return ProfileScreen(userId: userId);
      },
    ),
  ],
);
// Use in MaterialApp
MaterialApp.router(routerConfig: router);
```

GoRouter — Navigation Methods

```dart
// Navigate to a route
context.go('/details');           // Replaces entire stack
context.goNamed('details');       // By name

// Push onto stack (preserves back navigation)
context.push('/details');
context.pushNamed('details');

// Go back
context.pop();

// With query parameters
context.go('/search?q=flutter&page=1');
// Access in builder:
final query = state.uri.queryParameters['q']; // 'flutter'

// With extra data
context.go('/details', extra: myObject);
// Access:
final data = state.extra as MyObject;
```

GoRouter — Named Routes

```
final router = GoRouter(
  routes: [
    GoRoute(
      name: 'home',
      path: '/',
      builder: (context, state) => const HomeScreen(),
    ),
    GoRoute(
      name: 'profile',
      path: '/profile/:userId',
      builder: (context, state) {
        final userId = state.pathParameters['userId']!;
        return ProfileScreen(userId: userId);
      },
    ),
    GoRoute(
      name: 'search',
      path: '/search',
      builder: (context, state) {
        final q = state.uri.queryParameters['q'] ?? '';
        return SearchScreen(query: q);
      },
    ),
  ],
);
```

GoRouter — Named Routes

```
// Navigate by name with params
context.goNamed('profile', pathParameters: {'userId': '42'});
context.goNamed('search', queryParameters: {'q': 'flutter'});
```

Nested Navigation & Shell Routes

Shell routes let you share a layout (like a bottom nav bar) across multiple screens.

```
GoRouter(
  routes: [
    ShellRoute(
      builder: (context, state, child) => ScaffoldWithNavBar(child: child),
      routes: [
        GoRoute(path: '/', builder: (_, __) => const HomeTab()),
        GoRoute(path: '/search', builder: (_, __) => const SearchTab()),
        GoRoute(path: '/profile', builder: (_, __) => const ProfileTab()),
      ],
    ),
    GoRoute(path: '/login', builder: (_, __) => const LoginScreen()),
  ],
)
```

Key: The ShellRoute wraps tabs with a shared scaffold. Full-screen routes (like login) sit outside the shell.

Navigation Guards & Auth

Control access to routes based on authentication state.

```
final router = GoRouter(
  redirect: (context, state) {
    final isLoggedIn = AuthService.instance.isLoggedIn;
    final isLoginRoute = state.matchedLocation == '/login';

    if (!isLoggedIn && !isLoginRoute) return '/login';
    if (isLoggedIn && isLoginRoute) return '/';
    return null; // No redirect needed
  },
  routes: [
    GoRoute(path: '/', builder: (_, __) => const HomeScreen()),
    GoRoute(path: '/login', builder: (_, __) => const LoginScreen()),
    GoRoute(
      path: '/admin',
      builder: (_, __) => const AdminScreen(),
      // Per-route guard
      redirect: (context, state) {
        if (!AuthService.instance.isAdmin) return '/';
        return null;
      },
    ),
```

Deep Links (Android & iOS)

Deep links let users open your app from a URL.

Deep Links (Android & iOS)

Android Setup:

```
<!-- AndroidManifest.xml -->
<intent-filter>
  <action android:name="android.intent.action.VIEW" />
  <category android:name="android.intent.category.DEFAULT" />
  <category android:name="android.intent.category.BROWSABLE" />
  <data android:scheme="https" android:host="myapp.com" />
</intent-filter>
```

Deep Links (Android & iOS)

iOS Setup:

```
<!-- Info.plist -->
<key>CFBundleURLTypes</key>
<array>
  <dict>
    <key>CFBundleURLSchemes</key>
    <array><string>myapp</string></array>
  </dict>
</array>
```

> GoRouter handles deep links automatically — just define matching routes.

Adaptive Navigation Patterns

Different form factors need different navigation patterns.

Phone

- Bottom Navigation Bar
- Drawer (hamburger menu)
- Stack-based push/pop

Tablet

- Side rail / Navigation rail
- Master-detail (split view)

```
LayoutBuilder(
  builder: (context, constraints) {
    final isWide = constraints.maxWidth > 600;
    return isWide
      ? NavigationRail(
          destinations: [...],
          selectedIndex: _currentIndex,
          onDestinationSelected: _onTap,
        )
      : BottomNavigationBar(
          items: [...],
          currentIndex: _currentIndex,
          onTap: _onTap,
        );
  },
)
```

Mini Lab

Multi-Flow App with GoRouter

Objective: Build a multi-flow app with auth, tabs, and detail screens.

Requirements:

1. Login screen → redirects to home on success.
2. Home with bottom nav (3 tabs).
3. Detail screen pushed from tab content.
4. Auth guard: unauthenticated users see login.
5. Logout clears session and redirects.

Mini Lab: Step-by-Step

```
final router = GoRouter(
  redirect: (context, state) {
    final auth = AuthService.instance;
    final isLogin = state.matchedLocation == '/login';
    if (!auth.isLoggedIn && !isLogin) return '/login';
    if (auth.isLoggedIn && isLogin) return '/';
    return null;
  },
  routes: [
    GoRoute(path: '/login', builder: (_, __) => const LoginScreen()),
    ShellRoute(
      builder: (_, __, child) => MainScaffold(child: child),
      routes: [
        GoRoute(path: '/', builder: (_, __) => const HomeTab()),
        GoRoute(path: '/explore', builder: (_, __) => const ExploreTab()),
        GoRoute(path: '/profile', builder: (_, __) => const ProfileTab()),
      ],
    ),
    GoRoute(
      path: '/details/:id',
      builder: (_, state) {
        final id = state.pathParameters['id']!;
        return DetailScreen(id: id);
      },
    ),
  ],
);
```

Summary: Key Takeaways

Navigation Basics

- Routes are screens managed as a stack.
- Push adds, Pop removes.
- Declarative > Imperative for scalability.

GoRouter

- Official declarative router.
- Built-in deep links, guards, nested routes.
- Use `go` for replacement, `push` for stacking.

Advanced

- Shell routes for shared layouts.
- Auth guards via `redirect`.
- Deep links for external URLs.
- Adaptive patterns for phone/tablet.

Best Practices

- Define all routes in one place.
- Use named routes for readability.
- Handle back button on Android.
- Test navigation flows.

Thank You! 🙌

Questions?

"A router walks into a bar.
The bartender says, 'Where to?'
The router replies, 'I don't know, let me check my stack.'"