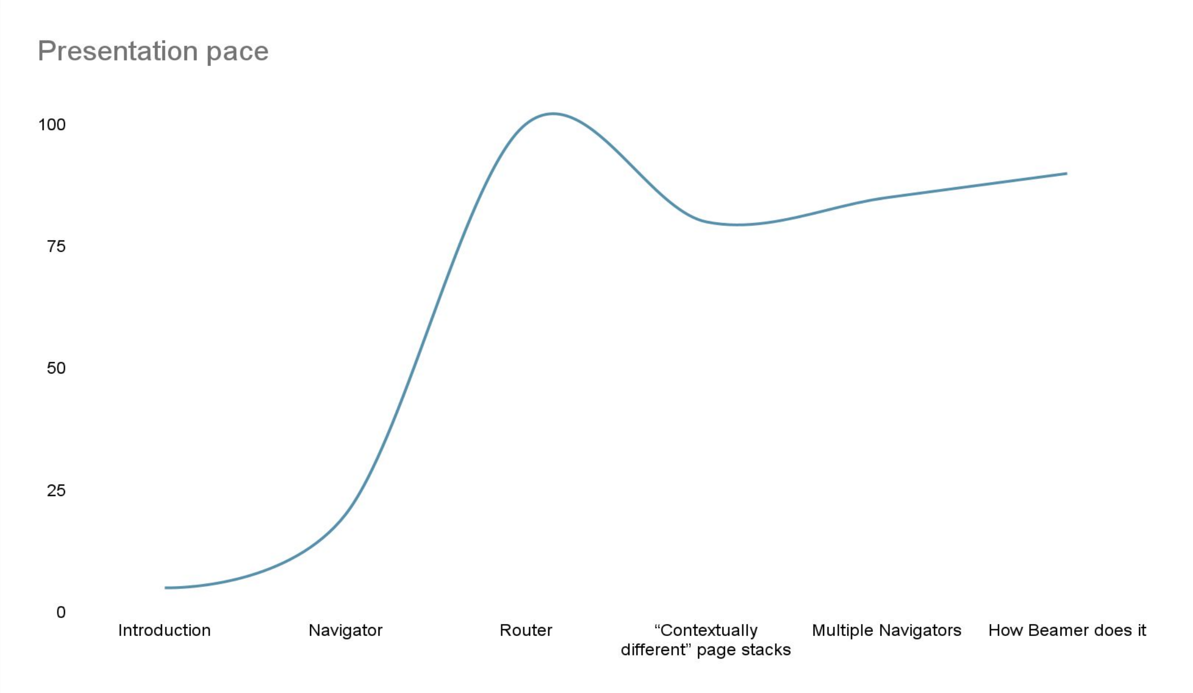
Router

Problems and solutions

Table of contents

1. Introduction: Whys and Hows
2. Navigator
3. Router
4. “Contextually different” page stacks
5. Multiple Navigators
6. How Beamer does it

Introduction



Introduction: Whys

- Complete integration with all platforms, specifically web
- Declarative API, more in line with how the rest of Flutter works

Introduction: Hows

- Complete integration with browsers
  - RouteInformationProvider
  - RouteInformationParser
  - currentConfiguration getter, etc.
- Declarative API, more in line with how the rest of Flutter works

Introduction: Hows

- Complete integration with browsers
  - RouteInformationProvider
  - RouteInformationParser
  - currentConfiguration getter, etc.
- Declarative API, more in line with how the rest of Flutter works
  - Navigator.pages API
  - Page stack is built by declaring a List of pages instead of pushing various pages
  - Natural deep-linking

Navigator

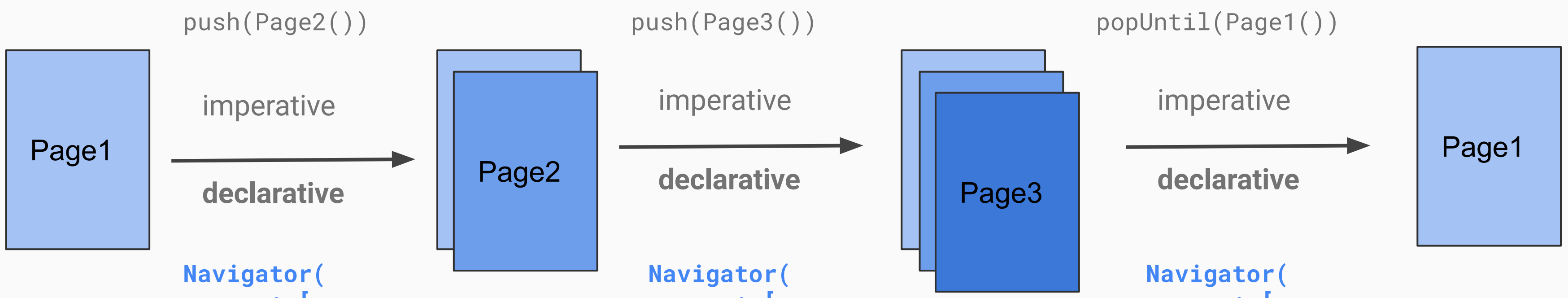
- a **Widget** that holds and manages a list of pages currently displayed (stacked) on the screen.
- Supports 2 *different* APIs; imperative and declarative
  - **Imperative**: Navigator.push, Navigator.pop, etc.
  - **Declarative**: rebuild Navigator with a new List of pages

Navigator

- a **Widget** that holds and manages a list of pages currently displayed (stacked) on the screen.
- Supports 2 *different* APIs; imperative and declarative
    - **Imperative**: Navigator.push, Navigator.pop, etc.
    - **Declarative**: rebuild Navigator with a new List of pages

“Every navigation event is deep-linking”

Navigator



```
Navigator(
 pages: [
  Page1(),
  Page2(),
 ],
)
```

```
Navigator(
 pages: [
  Page1(),
  Page2(),
  Page3(),
 ],
)
```

```
Navigator(
 pages: [
  Page1(),
 ],
)
```

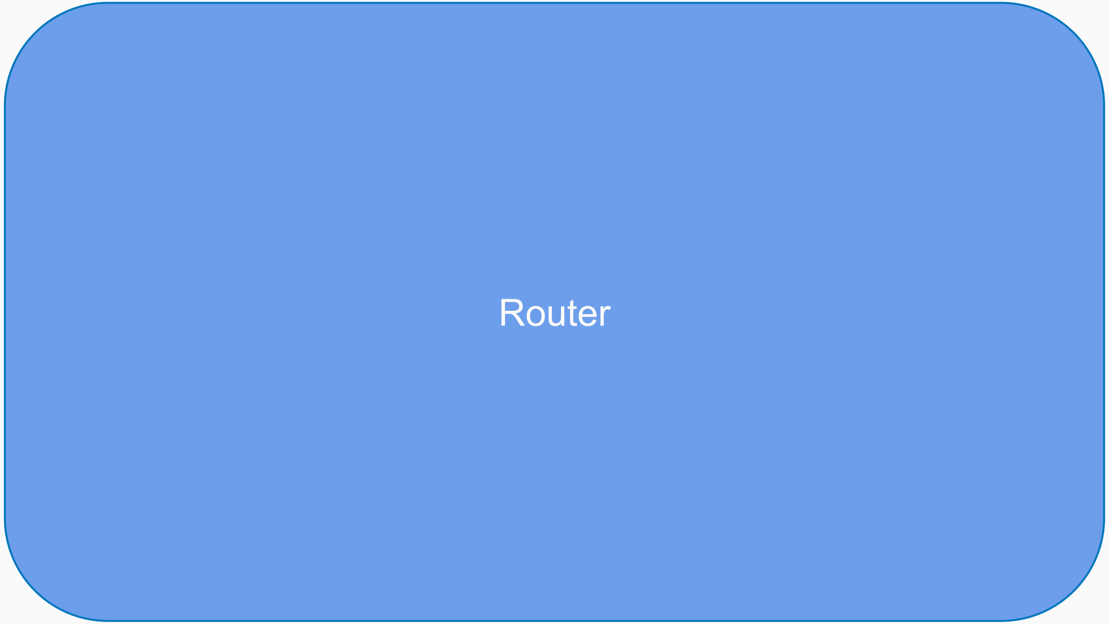
Navigator

How and where to rebuild it?

- Regular setState in your StatefulWidget that builds Navigator
- **Via Router**

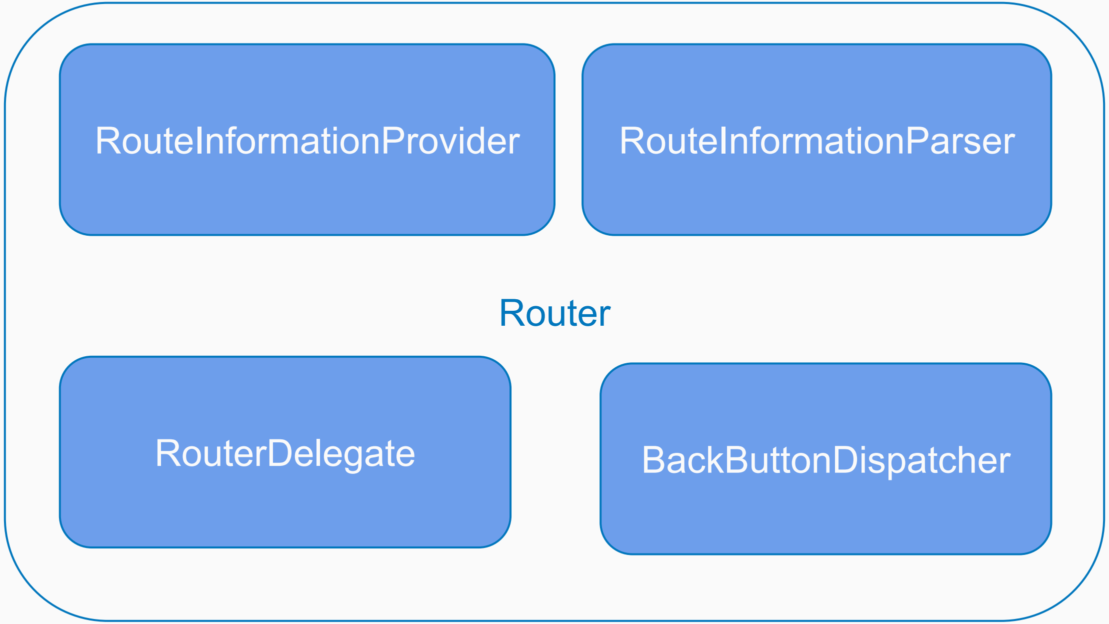
Important: When using the declarative Pages API, *Navigator.onPopPage* must also be implemented to define how state (which governs the selection of pages) should be updated after a BackButton is pressed.

Router



Router

Router



RouteInformationProvider

RouteInformationParser

Router

RouterDelegate

BackButtonDispatcher

Router

- A **Widget** that uses its components to
  - Get a piece of routing information from and to platform (**RouteInformationProvider**, usually *PlatformRouteInformationProvider*)
  - Parse the routing information into an object that it wants to work with internally (**RouteInformationParser**)
  - Build a Navigator widget and notifies of any changes in routing (**RouterDelegate**)
  - Handle the (android) back button (**BackButtonDispatcher**)

Router

How to use it? *(in short)*

- MaterialApp.router
- Decide how to represent routes and state
- Implement RouteInformationParser
    - Override parseRouteInformation
    - Override restoreRouteInformation
- Implement RouterDelegate
    - Override setNewRoutePath
    - Override build
        - return a Navigator* populated with a list of pages reflecting the routing state
        - Make sure to update routing state in Navigator.onPopPage
    - Override currentConfiguration
- Optionally override BackButtonDispatcher

** any widget can be returned*

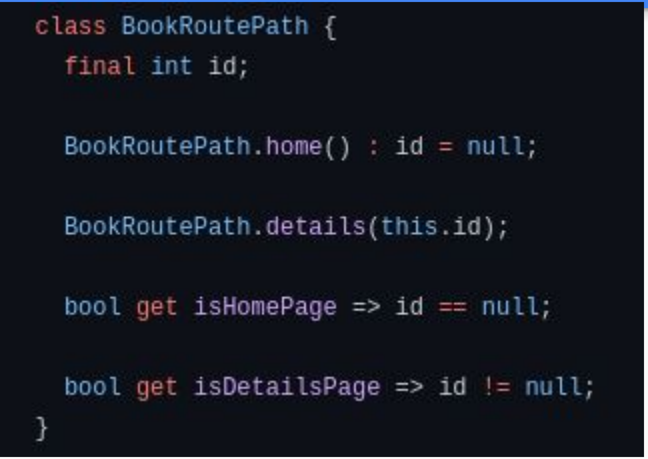
Router

How to use it?

- MaterialApp.router
- Decide how to represent the routing state
- Implement RouteInformationParser
    - Override parseRouteInformation
    - Override restoreRouteInformation
- Implement RouterDelegate
    - Override setNewRoutePath
    - Override build
        - return a Navigator* populated with a list of pages reflecting the routing state
        - Make sure to update routing state in Navigator.onPopPage
    - Override currentConfiguration
- Optionally override BackButtonDispatcher

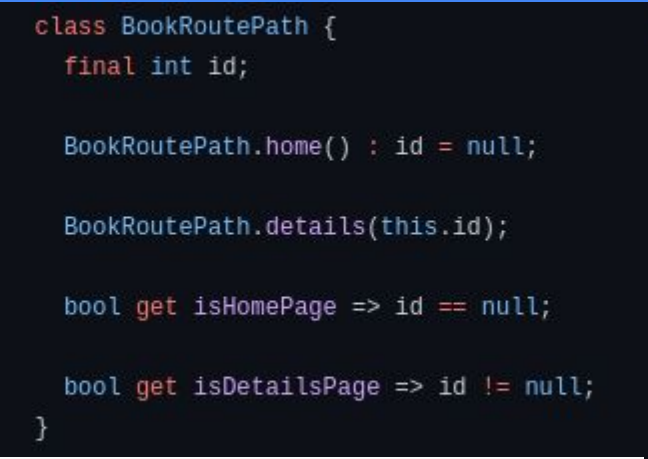
** any widget can be returned*

Router

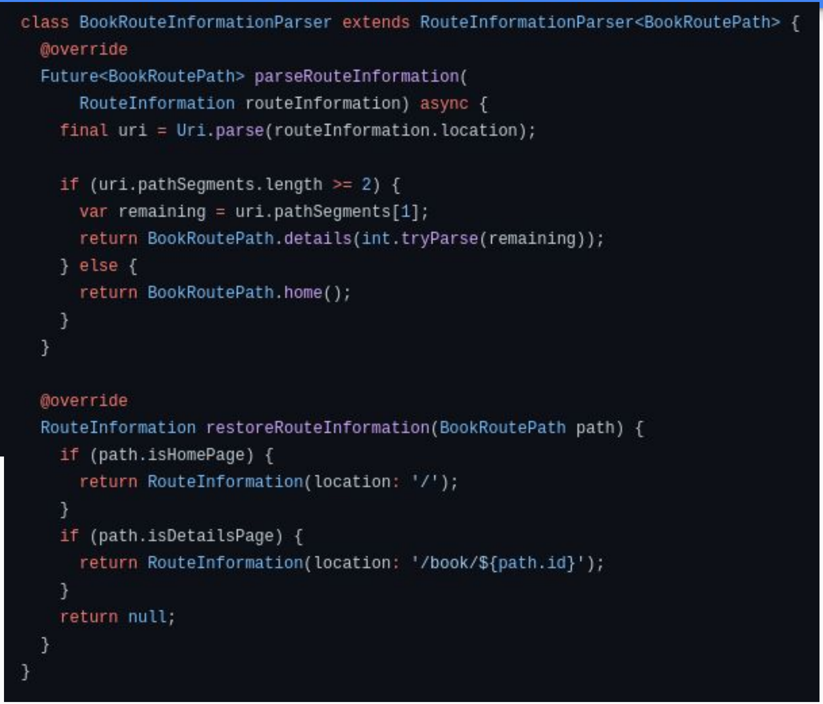


```
class BookRoutePath {
  final int id;

  BookRoutePath.home() : id = null;

  BookRoutePath.details(this.id);

  bool get isHomePage => id == null;

  bool get isDetailsPage => id != null;
}
```

Router

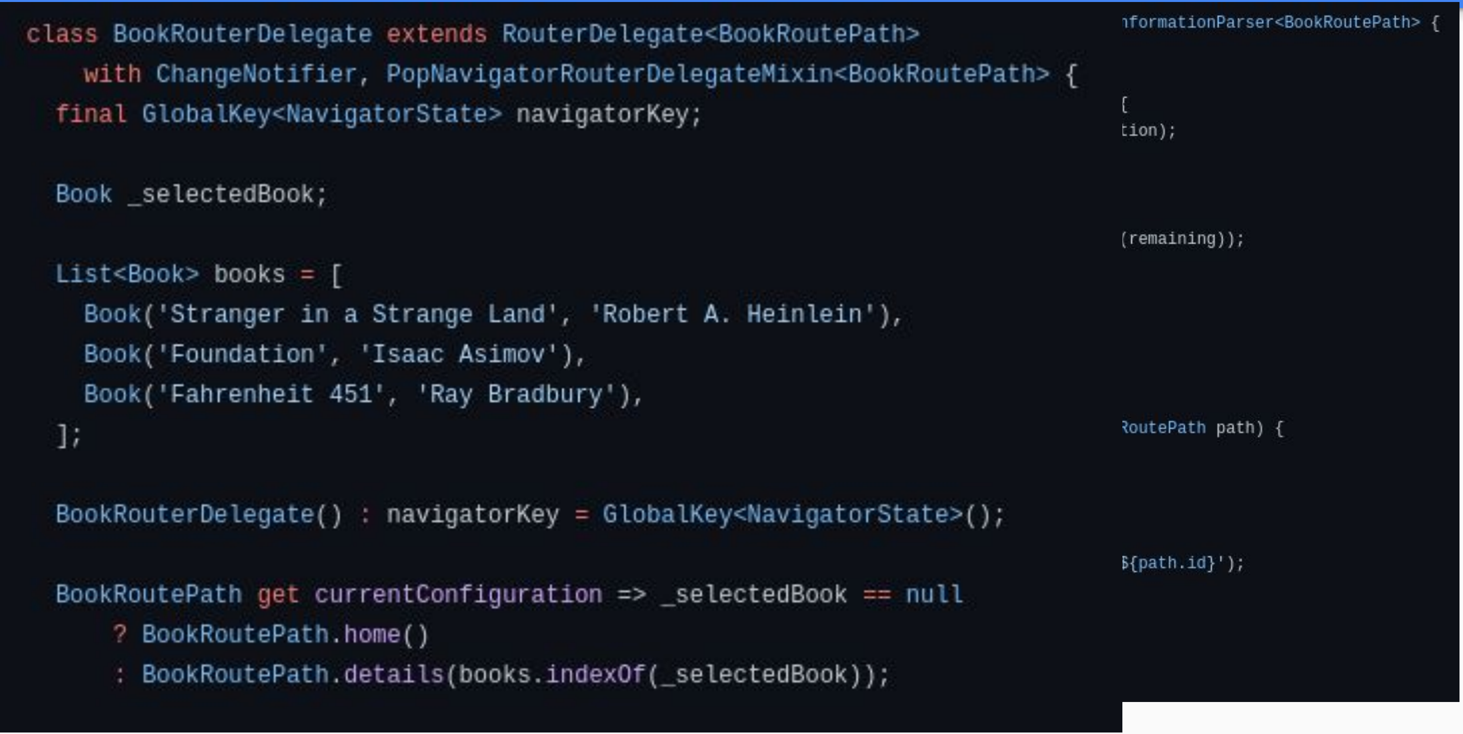


```
class BookRoutePath {
  final int id;

  BookRoutePath.home() : id = null;

  BookRoutePath.details(this.id);

  bool get isHomePage => id == null;

  bool get isDetailsPage => id != null;
}
```



```
class BookRouteInformationParser extends RouteInformationParser<BookRoutePath> {
  @override
  Future<BookRoutePath> parseRouteInformation(
      RouteInformation routeInformation) async {
    final uri = Uri.parse(routeInformation.location);

    if (uri.pathSegments.length >= 2) {
      var remaining = uri.pathSegments[1];
      return BookRoutePath.details(int.tryParse(remaining));
    } else {
      return BookRoutePath.home();
    }
  }

  @override
  RouteInformation restoreRouteInformation(BookRoutePath path) {
    if (path.isHomePage) {
      return RouteInformation(location: '/');
    }
    if (path.isDetailsPage) {
      return RouteInformation(location: '/book/${path.id}');
    }
    return null;
  }
}
```

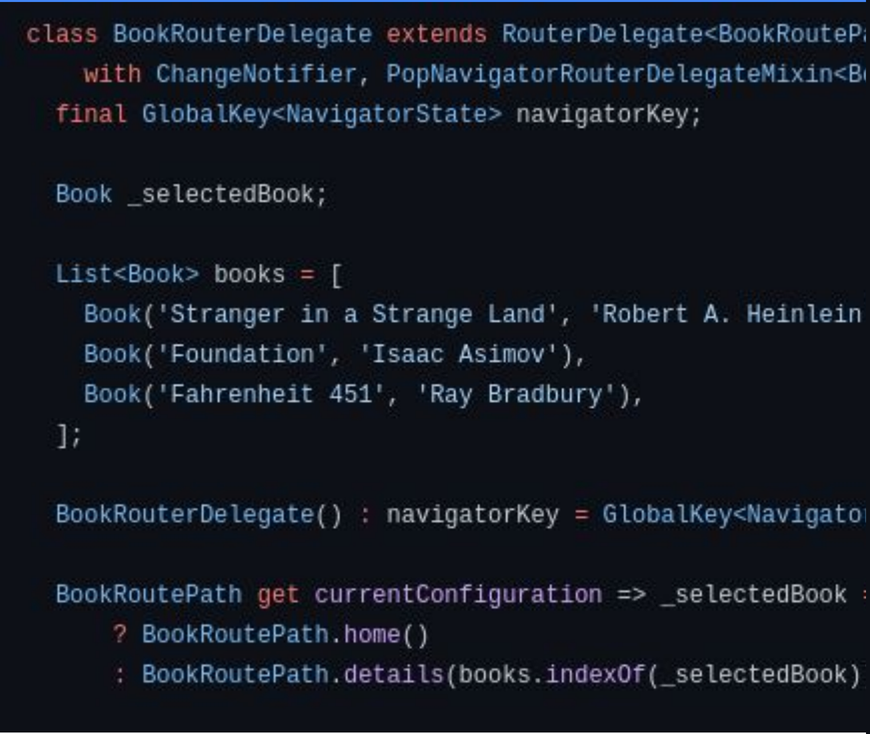
Router



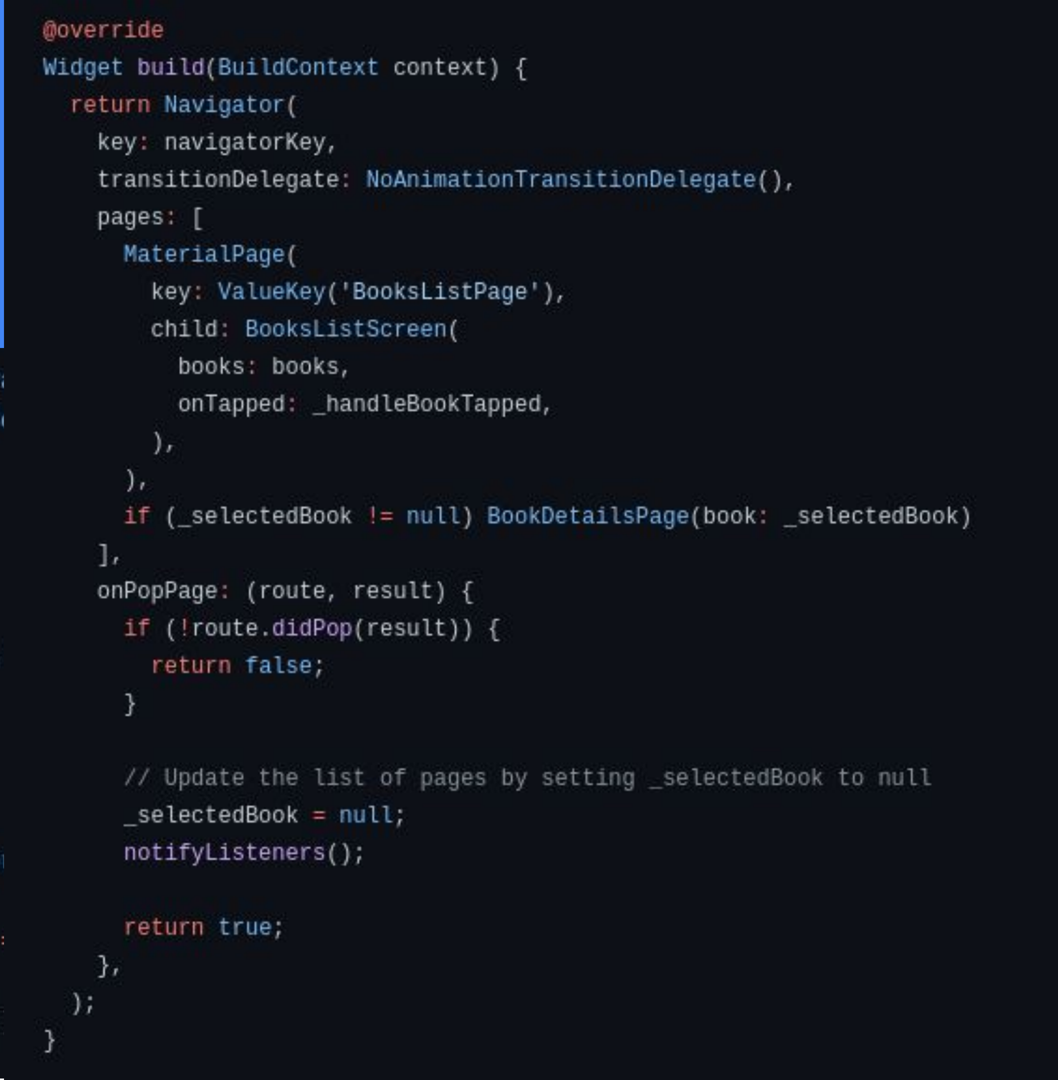
```
class BookRouterDelegate extends RouterDelegate<BookRoutePath>
    with ChangeNotifier, PopNavigatorRouterDelegateMixin<BookRoutePath> {
  final GlobalKey<NavigatorState> navigatorKey;

  Book _selectedBook;

  List<Book> books = [
    Book('Stranger in a Strange Land', 'Robert A. Heinlein'),
    Book('Foundation', 'Isaac Asimov'),
    Book('Fahrenheit 451', 'Ray Bradbury'),
  ];

  BookRouterDelegate() : navigatorKey = GlobalKey<NavigatorState>();

  BookRoutePath get currentConfiguration => _selectedBook == null
      ? BookRoutePath.home()
      : BookRoutePath.details(books.indexOf(_selectedBook));
```

```
iformationParser<BookRoutePath> {

{
tion);

(remaining));

RoutePath path) {

${path.id}');
```

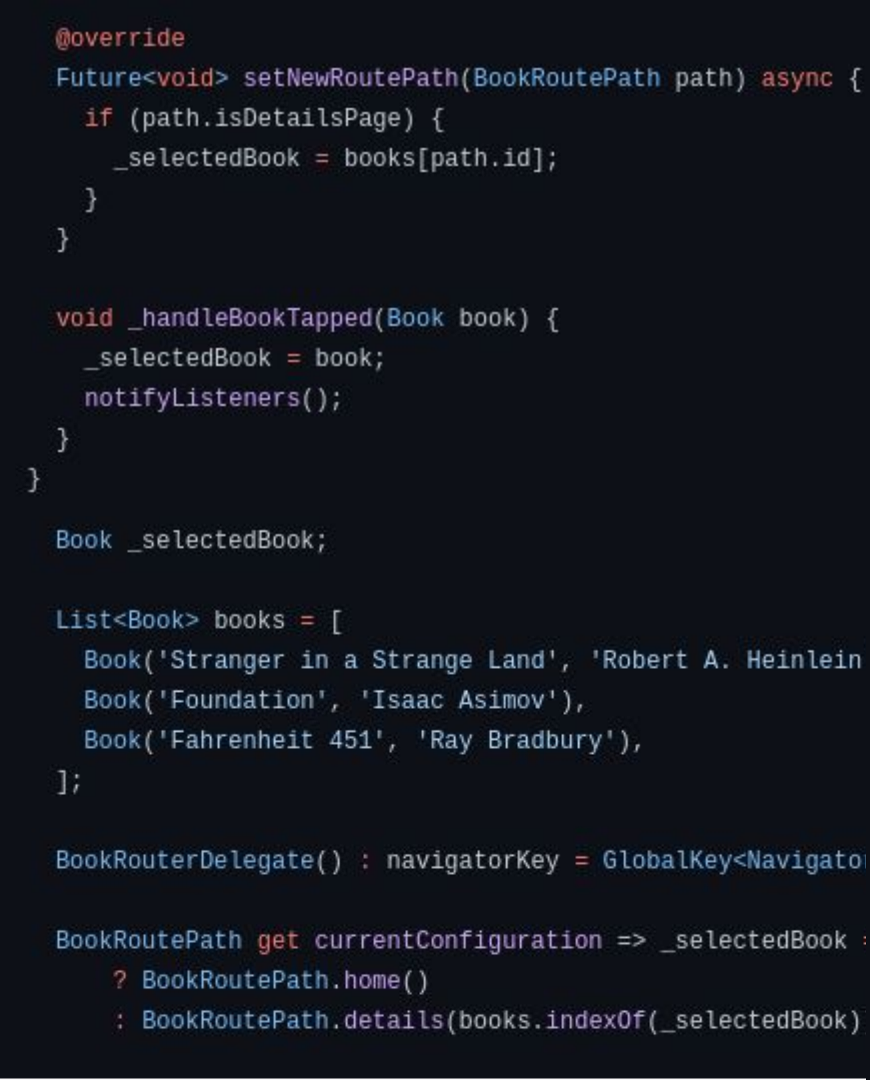
Router



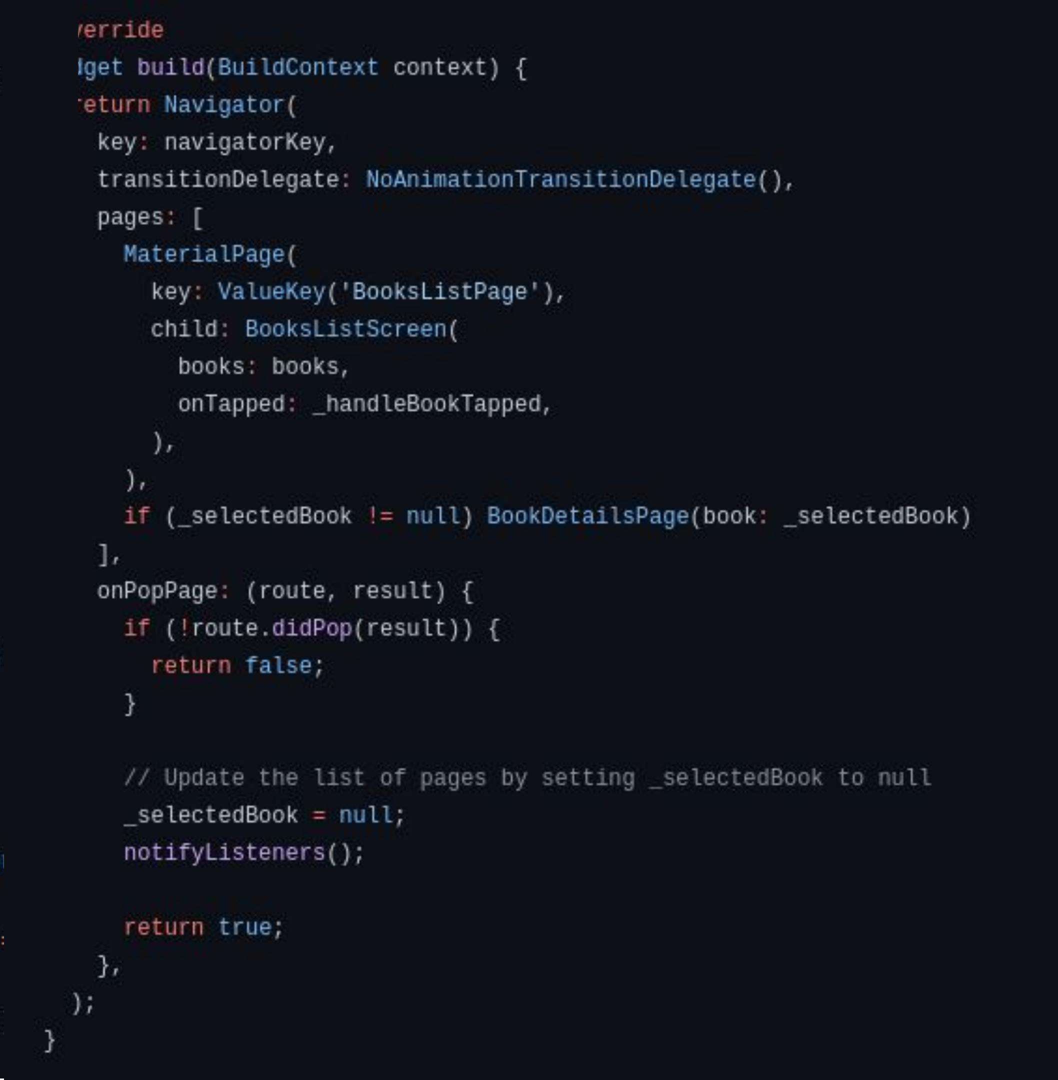
```
class BookRouterDelegate extends RouterDelegate<BookRouteP
    with ChangeNotifier, PopNavigatorRouterDelegateMixin<B
  final GlobalKey<NavigatorState> navigatorKey;

  Book _selectedBook;

  List<Book> books = [
    Book('Stranger in a Strange Land', 'Robert A. Heinlein
    Book('Foundation', 'Isaac Asimov'),
    Book('Fahrenheit 451', 'Ray Bradbury'),
  ];

  BookRouterDelegate() : navigatorKey = GlobalKey<Navigato

  BookRoutePath get currentConfiguration => _selectedBook
      ? BookRoutePath.home()
      : BookRoutePath.details(books.indexOf(_selectedBook)
```



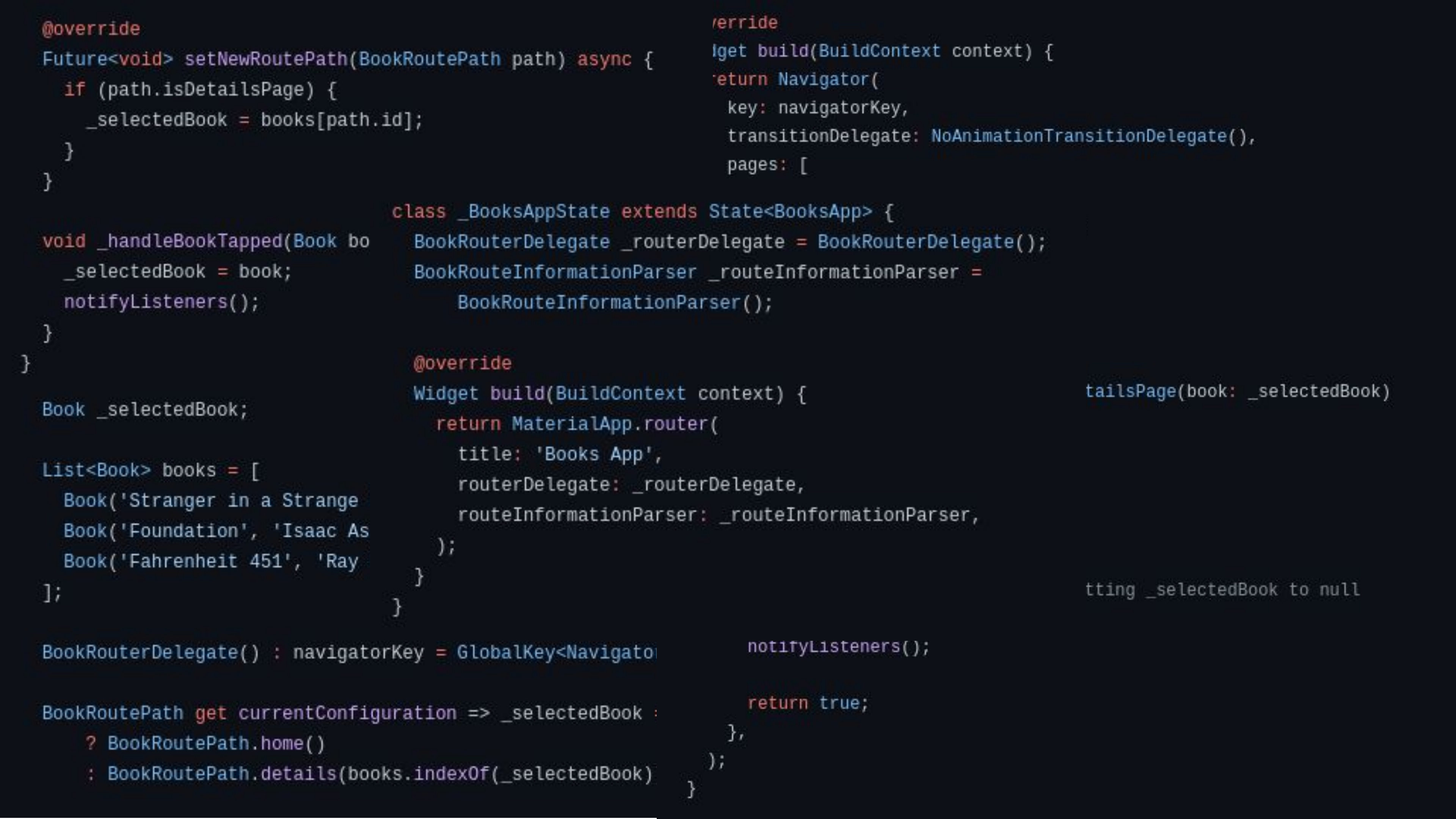
```
@override
Widget build(BuildContext context) {
  return Navigator(
    key: navigatorKey,
    transitionDelegate: NoAnimationTransitionDelegate(),
    pages: [
      MaterialPage(
        key: ValueKey('BooksListPage'),
        child: BooksListScreen(
          books: books,
          onTapped: _handleBookTapped,
        ),
      ),
      if (_selectedBook != null) BookDetailsPage(book: _selectedBook)
    ],
    onPopPage: (route, result) {
      if (!route.didPop(result)) {
        return false;
      }

      // Update the list of pages by setting _selectedBook to null
      _selectedBook = null;
      notifyListeners();

      return true;
    },
  );
}
```



```
  @override
  Future<void> setNewRoutePath(BookRoutePath path) async {
    if (path.isDetailsPage) {
      _selectedBook = books[path.id];
    }
  }

  void _handleBookTapped(Book book) {
    _selectedBook = book;
    notifyListeners();
  }
}

  Book _selectedBook;

  List<Book> books = [
    Book('Stranger in a Strange Land', 'Robert A. Heinlein
    Book('Foundation', 'Isaac Asimov'),
    Book('Fahrenheit 451', 'Ray Bradbury'),
  ];

  BookRouterDelegate() : navigatorKey = GlobalKey<Navigato

  BookRoutePath get currentConfiguration => _selectedBook
      ? BookRoutePath.home()
      : BookRoutePath.details(books.indexOf(_selectedBook)
```



```
  verride
  get build(BuildContext context) {
  eturn Navigator(
    key: navigatorKey,
    transitionDelegate: NoAnimationTransitionDelegate(),
    pages: [
      MaterialPage(
        key: ValueKey('BooksListPage'),
        child: BooksListScreen(
          books: books,
          onTapped: _handleBookTapped,
        ),
      ),
      if (_selectedBook != null) BookDetailsPage(book: _selectedBook)
    ],
    onPopPage: (route, result) {
      if (!route.didPop(result)) {
        return false;
      }

      // Update the list of pages by setting _selectedBook to null
      _selectedBook = null;
      notifyListeners();

      return true;
    },
  );
}
```



```
  @override
  Future<void> setNewRoutePath(BookRoutePath path) async {
    if (path.isDetailsPage) {
      _selectedBook = books[path.id];
    }
  }

  void _handleBookTapped(Book bo
    _selectedBook = book;
    notifyListeners();
  }
}
```

```
/erride
lget build(BuildContext context) {
'eturn Navigator(
  key: navigatorKey,
  transitionDelegate: NoAnimationTransitionDelegate(),
  pages: [
```

```
class _BooksAppState extends State<BooksApp> {
  BookRouterDelegate _routerDelegate = BookRouterDelegate();
  BookRouteInformationParser _routeInformationParser =
      BookRouteInformationParser();

  @override
  Widget build(BuildContext context) {
    return MaterialApp.router(
      title: 'Books App',
      routerDelegate: _routerDelegate,
      routeInformationParser: _routeInformationParser,
    );
  }
}
```

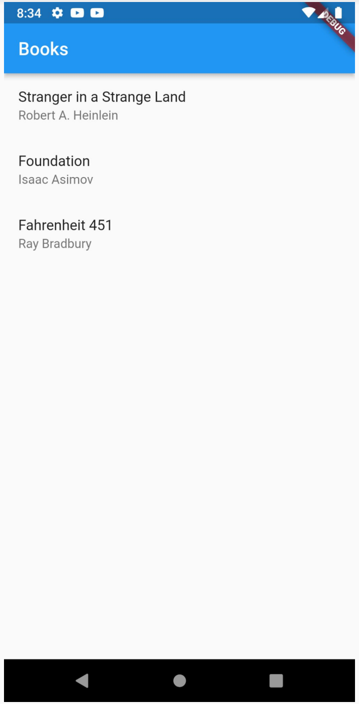
```
  Book _selectedBook;

  List<Book> books = [
    Book('Stranger in a Strange
    Book('Foundation', 'Isaac As
    Book('Fahrenheit 451', 'Ray
  ];
```

```
tailsPage(book: _selectedBook)

tting _selectedBook to null
```

```
  BookRouterDelegate() : navigatorKey = GlobalKey<Navigato

  BookRoutePath get currentConfiguration => _selectedBook :
      ? BookRoutePath.home()
      : BookRoutePath.details(books.indexOf(_selectedBook)
```

```
      notifyListeners();

      return true;
    },
  );
}
```

Router

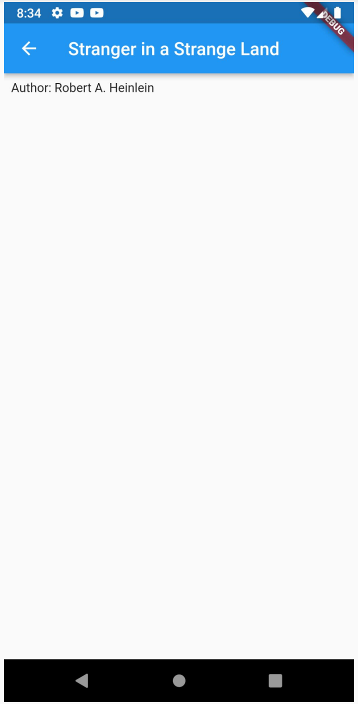


Books

Stranger in a Strange Land
Robert A. Heinlein

Foundation
Isaac Asimov

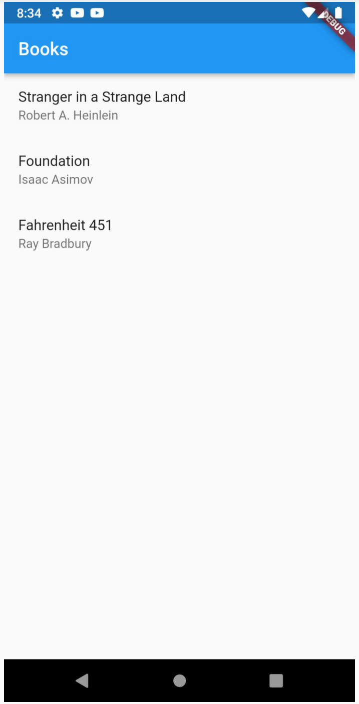
Fahrenheit 451
Ray Bradbury



Stranger in a Strange Land

Author: Robert A. Heinlein

Router



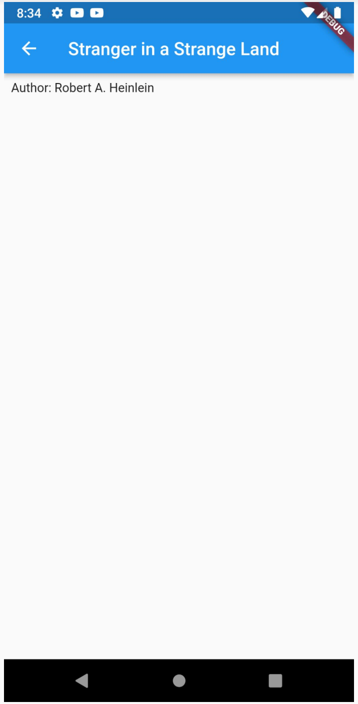
8:34

Books

Stranger in a Strange Land
Robert A. Heinlein

Foundation
Isaac Asimov

Fahrenheit 451
Ray Bradbury



8:34

Stranger in a Strange Land

Author: Robert A. Heinlein

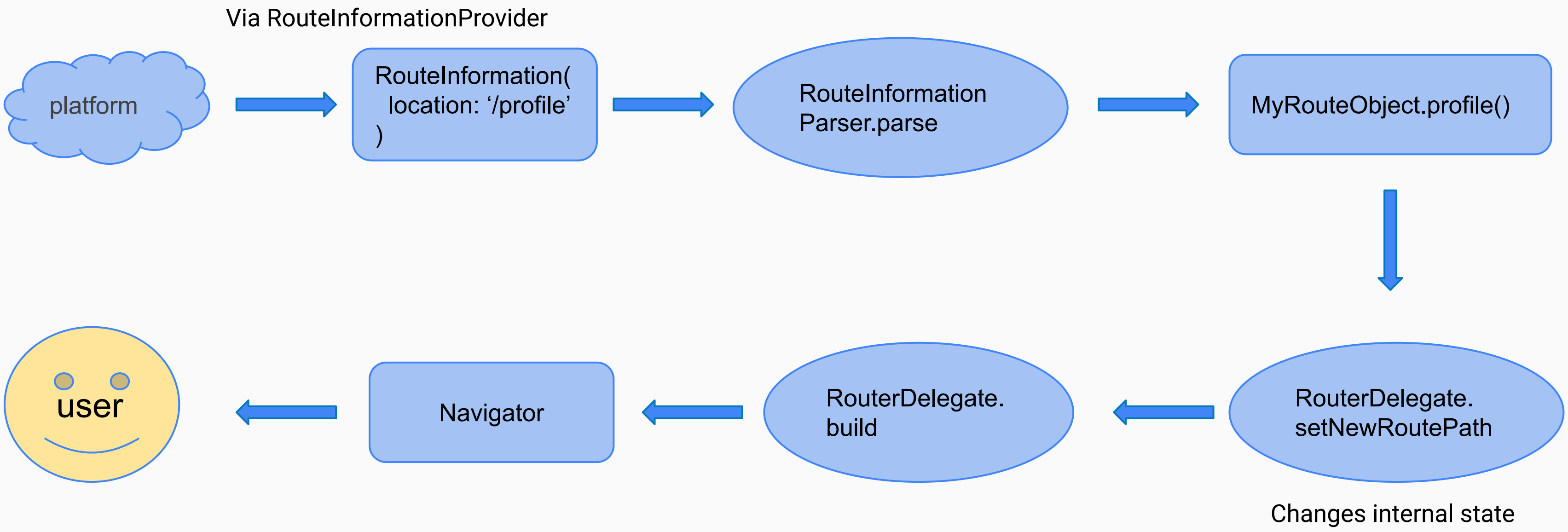
Router

A lot of boilerplate we have seen is due to **flexibility** and **power** of Router. We really can do almost anything we want, and that **is** amazing.

Various packages try to reduce the complexity of using the Router API. Some of the most popular are (alphabetically):

- auto_route
- beamer
- go_router
- routemaster
- vrouter

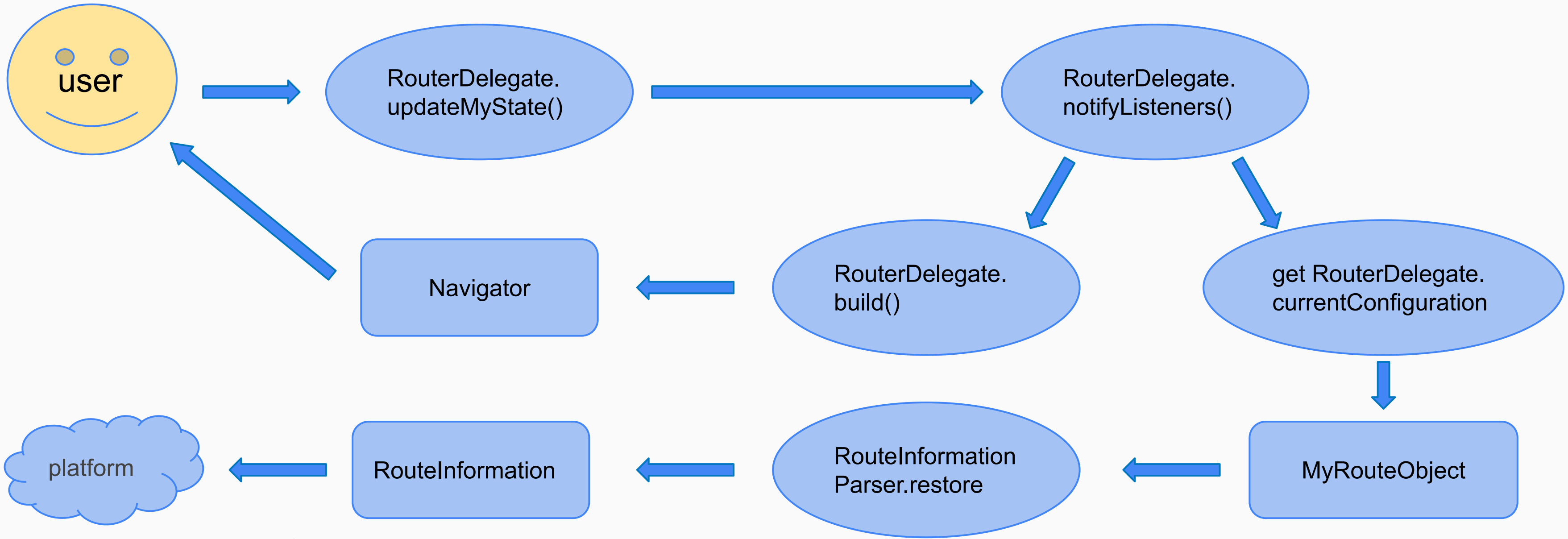
Router



Via RouteInformationProvider

Changes internal state

Router



user

RouterDelegate.
updateMyState()

RouterDelegate.
notifyListeners()

Navigator

RouterDelegate.
build()

get RouterDelegate.
currentConfiguration

platform

RouteInformation

RouteInformation
Parser.restore

MyRouteObject

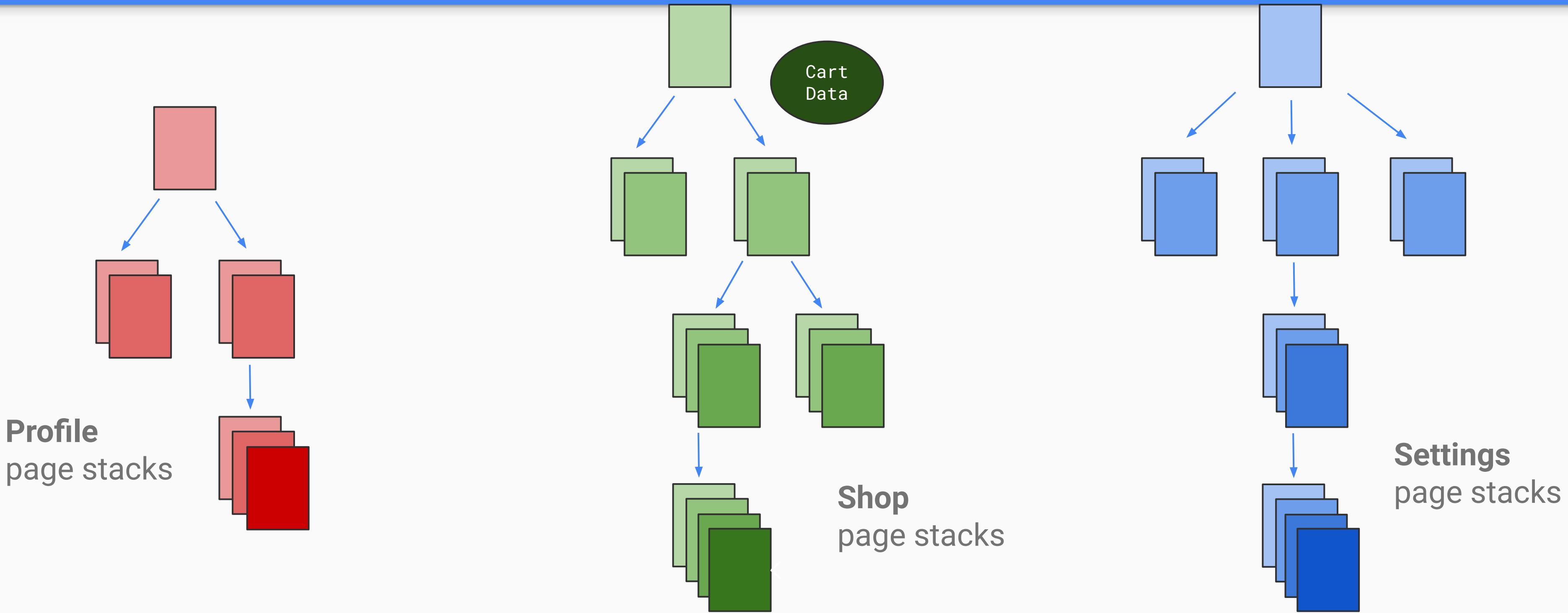
Via RouteInformationProvider

Router

Important thing to notice is that Router is responsible for building a **single** Navigator. Some natural questions arise:

- How to deal with **wide** (“*contextually different*”) **and deep** *page stacks*?
- How to implement tabbed navigation and keep the state of each tab’s stack of pages?

“Contextually different” page stacks



Profile
page stacks

Shop
page stacks

Settings
page stacks

“Contextually different” page stacks

In order to provider Cart data to all pages in a shop stack, we have to provide it or at least create it above Navigator. What if user never enters shop?

We would like to have some special builders for each “contextually different” stack of pages, where we can bind and provide data to just specific pages.

Multiple Navigators

So far, we've seen all the possible page stacks are built by a single Navigator. In the use case of tabbed navigation (e.g. Profile in first bottom tab and Shop in second), we have 2 options:

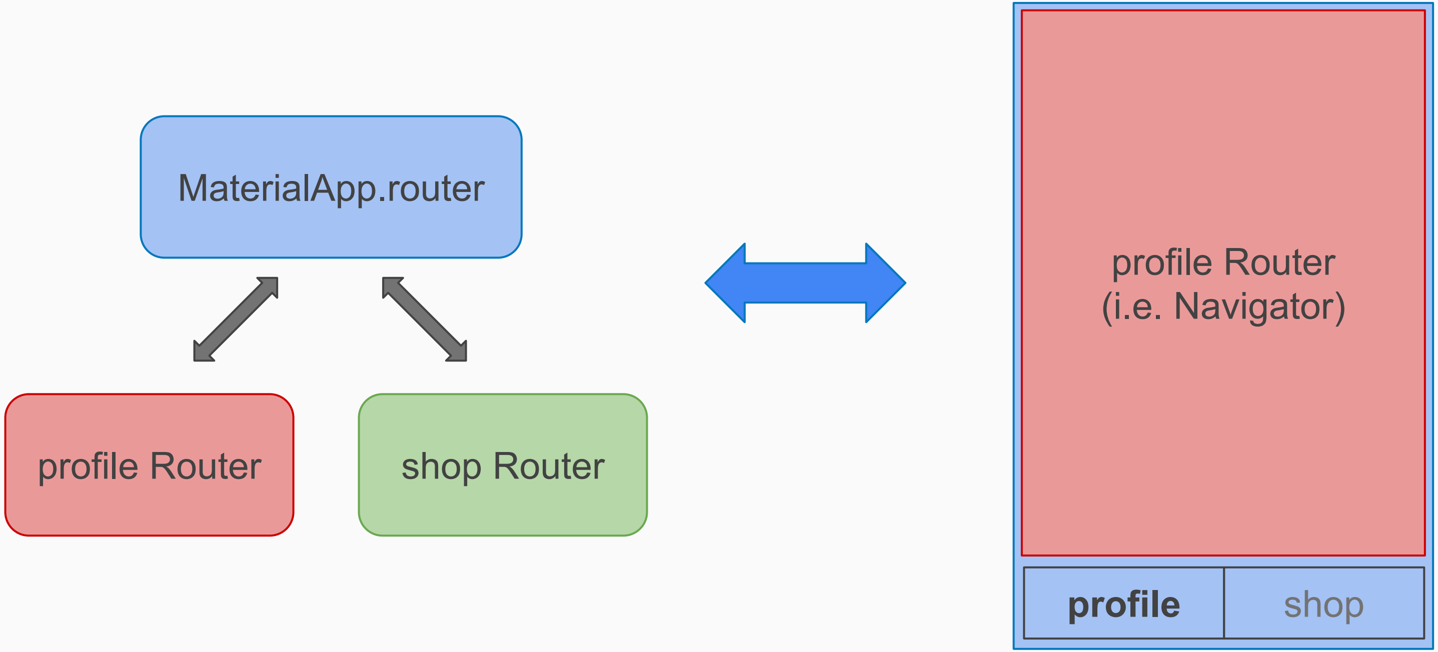
- Use a single Navigator for both tabs
    - Means that navigation to shop from somewhere deep in the profile will completely remove all profile pages and they will need to be created again if we were to come back
- **Build a Navigator in each tab**
    - Means that each tab's Navigator can independently rebuilt so we can keep the state of other tabs, especially if using something like IndexedStack

Multiple Navigators

How to create multiple Navigators while using Router API and how to keep them all in sync?

- ➔ As we learned, every Router builds a single Navigator so it would be natural to create nested Routers below MaterialApp.router

Multiple Navigators



MaterialApp.router

profile Router

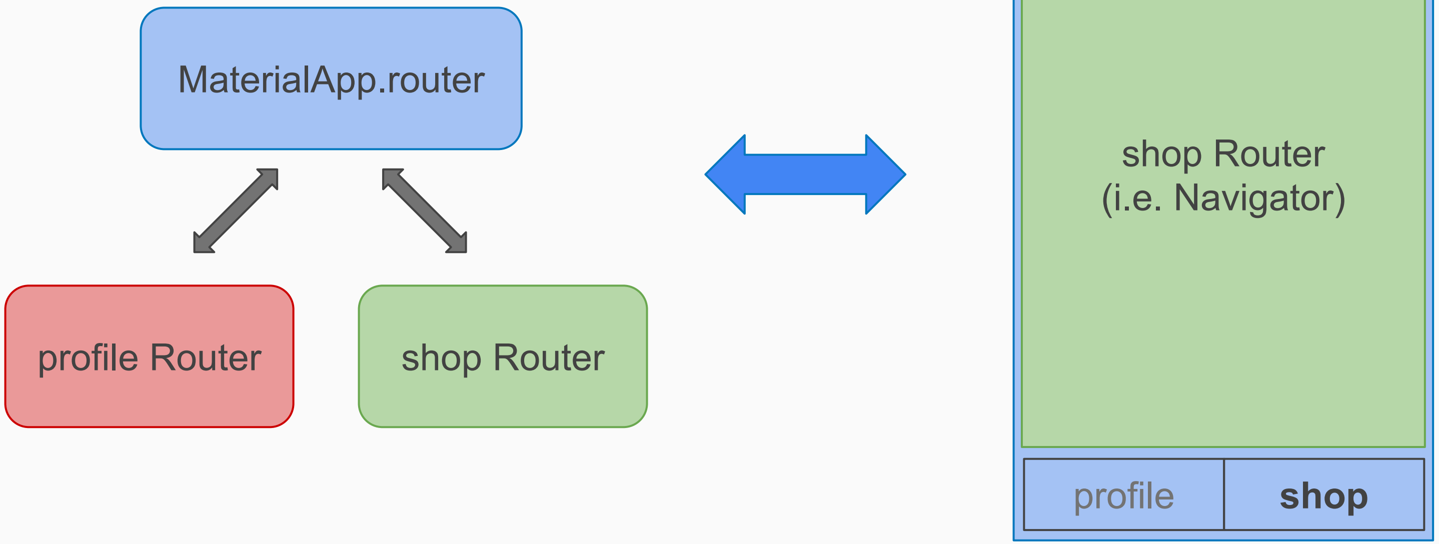
shop Router

profile Router
(i.e. Navigator)

profile

shop

Multiple Navigators



MaterialApp.router

profile Router

shop Router

shop Router
(i.e. Navigator)

profile

shop

Multiple Navigators

How to sync all the Routers in the app?

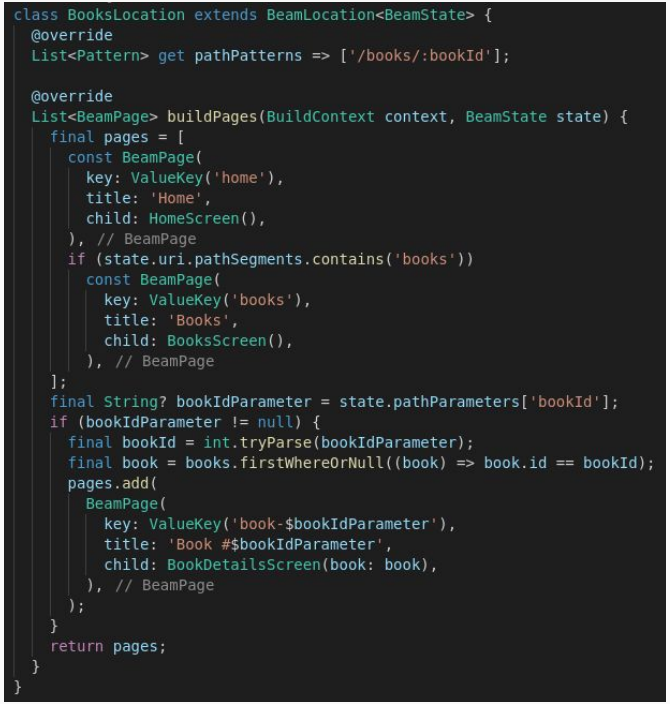
Multiple Navigators

How to sync all the Routers in the app?

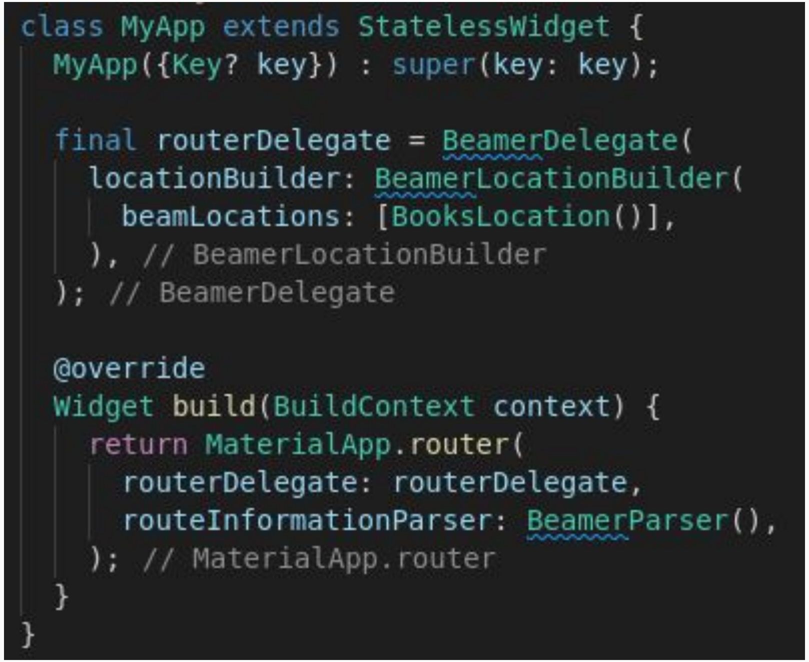
How Beamer does it

- Provides implementation for RouteInformationParser
- Provides implementation for RouterDelegate
- Focuses on contextually different page stacks whose “builders” are called **BeamLocations** that user can extend to define specific page stacks
- Achieves arbitrary nested navigation by letting the user insert **Beamer** widget anywhere in the app where nested Navigator should be built. (Beamer widget is essentially a Router widget that communicates with its parent Router)

How Beamer does it

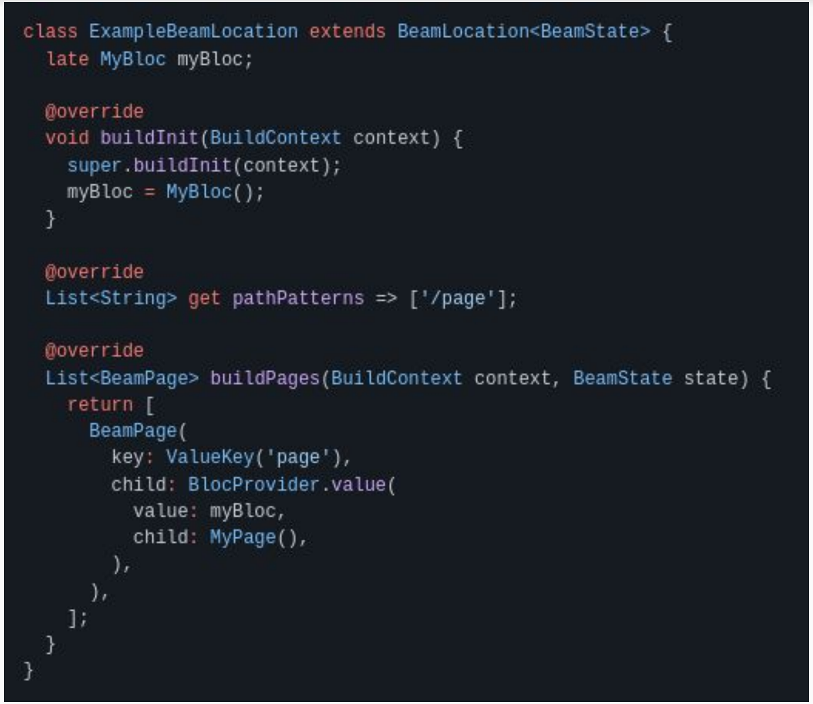


```dart
class BooksLocation extends BeamLocation<BeamState> {
  @override
  List<Pattern> get pathPatterns => ['/books/:bookId'];

  @override
  List<BeamPage> buildPages(BuildContext context, BeamState state) {
    final pages = [
      const BeamPage(
        key: ValueKey('home'),
        title: 'Home',
        child: HomeScreen(),
      ), // BeamPage
      if (state.uri.pathSegments.contains('books'))
        const BeamPage(
          key: ValueKey('books'),
          title: 'Books',
          child: BooksScreen(),
        ), // BeamPage
    ];
    final String? bookIdParameter = state.pathParameters['bookId'];
    if (bookIdParameter != null) {
      final bookId = int.tryParse(bookIdParameter);
      final book = books.firstWhereOrNull((book) => book.id == bookId);
      pages.add(
        BeamPage(
          key: ValueKey('book-$bookIdParameter'),
          title: 'Book #$bookIdParameter',
          child: BookDetailsScreen(book: book),
        ), // BeamPage
      );
    }
    return pages;
  }
}
```



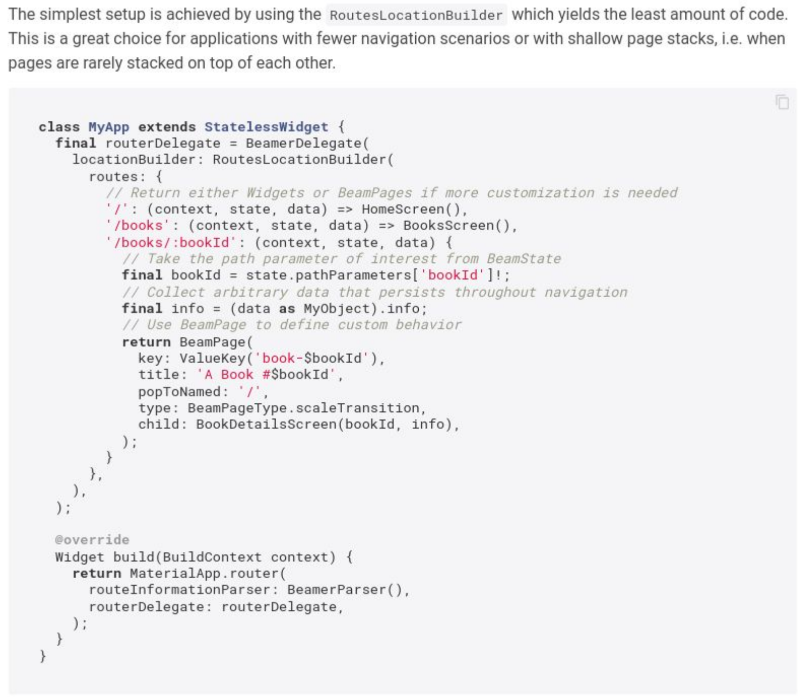
```dart
class MyApp extends StatelessWidget {
  MyApp({Key? key}) : super(key: key);

  final routerDelegate = BeamerDelegate(
    locationBuilder: BeamerLocationBuilder(
      beamLocations: [BooksLocation()],
    ), // BeamerLocationBuilder
  ); // BeamerDelegate

  @override
  Widget build(BuildContext context) {
    return MaterialApp.router(
      routerDelegate: routerDelegate,
      routeInformationParser: BeamerParser(),
    ); // MaterialApp.router
  }
}
```

How Beamer does it



```
class ExampleBeamLocation extends BeamLocation<BeamState> {
  late MyBloc myBloc;

  @override
  void buildInit(BuildContext context) {
    super.buildInit(context);
    myBloc = MyBloc();
  }

  @override
  List<String> get pathPatterns => ['/page'];

  @override
  List<BeamPage> buildPages(BuildContext context, BeamState state) {
    return [
      BeamPage(
        key: ValueKey('page'),
        child: BlocProvider.value(
          value: myBloc,
          child: MyPage(),
        ),
      ),
    ];
  }
}
```

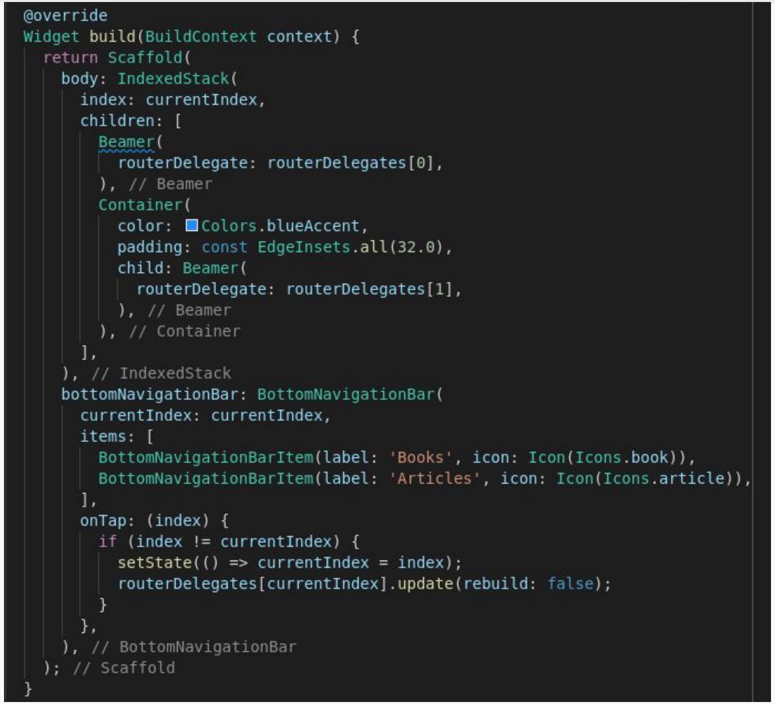
How Beamer does it



The simplest setup is achieved by using the `RoutesLocationBuilder` which yields the least amount of code. This is a great choice for applications with fewer navigation scenarios or with shallow page stacks, i.e. when pages are rarely stacked on top of each other.

```
class MyApp extends StatelessWidget {
  final routerDelegate = BeamerDelegate(
    locationBuilder: RoutesLocationBuilder(
      routes: {
        // Return either Widgets or BeamPages if more customization is needed
        '/': (context, state, data) => HomeScreen(),
        '/books': (context, state, data) => BooksScreen(),
        '/books/:bookId': (context, state, data) {
          // Take the path parameter of interest from BeamState
          final bookId = state.pathParameters['bookId']!;
          // Collect arbitrary data that persists throughout navigation
          final info = (data as MyObject).info;
          // Use BeamPage to define custom behavior
          return BeamPage(
            key: ValueKey('book-$bookId'),
            title: 'A Book #$bookId',
            popToNamed: '/',
            type: BeamPageType.scaleTransition,
            child: BookDetailsScreen(bookId, info),
          );
        }
      },
    ),
  );

  @override
  Widget build(BuildContext context) {
    return MaterialApp.router(
      routeInformationParser: BeamerParser(),
      routerDelegate: routerDelegate,
    );
  }
}
```

How Beamer does it



```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: IndexedStack(
      index: currentIndex,
      children: [
        Beamer(
          routerDelegate: routerDelegates[0],
        ), // Beamer
        Container(
          color: Colors.blueAccent,
          padding: const EdgeInsets.all(32.0),
          child: Beamer(
            routerDelegate: routerDelegates[1],
          ), // Beamer
        ), // Container
      ],
    ), // IndexedStack
    bottomNavigationBar: BottomNavigationBar(
      currentIndex: currentIndex,
      items: [
        BottomNavigationBarItem(label: 'Books', icon: Icon(Icons.book)),
        BottomNavigationBarItem(label: 'Articles', icon: Icon(Icons.article)),
      ],
      onTap: (index) {
        if (index != currentIndex) {
          setState(() => currentIndex = index);
          routerDelegates[currentIndex].update(rebuild: false);
        }
      },
    ), // BottomNavigationBar
  ); // Scaffold
}
```

Sources

- https://api.flutter.dev/flutter/widgets/Navigator-class.html
- https://api.flutter.dev/flutter/widgets/Router-class.html
- https://medium.com/flutter/learning-flutters-new-navigation-and-routing-system-7c9068155ade
- https://github.com/slovnicki/flutter-festival-london
- https://pub.dev/packages/beamer

Thanks!