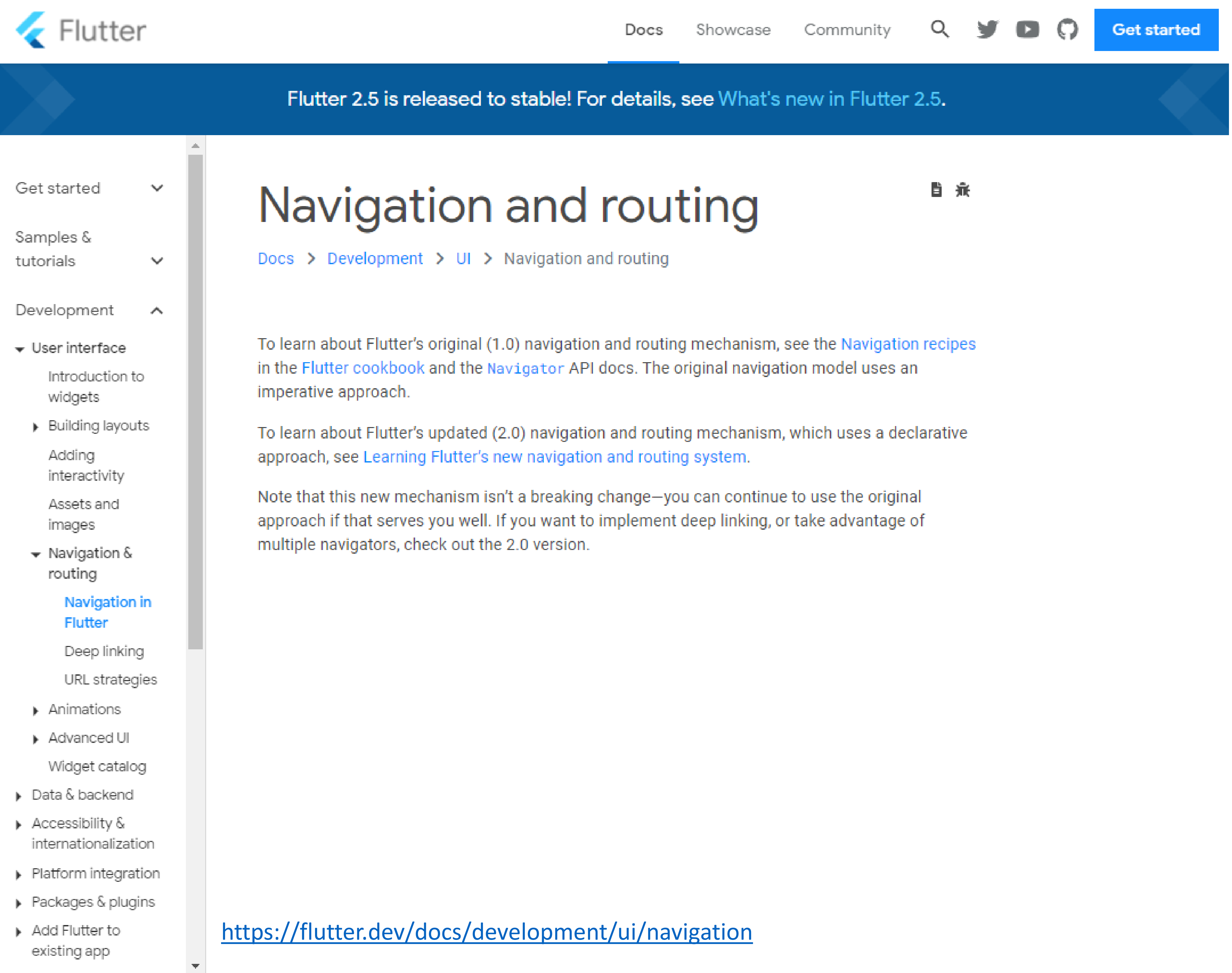
Navigation

universidade de aveiro



Flutter

Docs Showcase Community Get started

Flutter 2.5 is released to stable! For details, see What's new in Flutter 2.5.

Navigation and routing

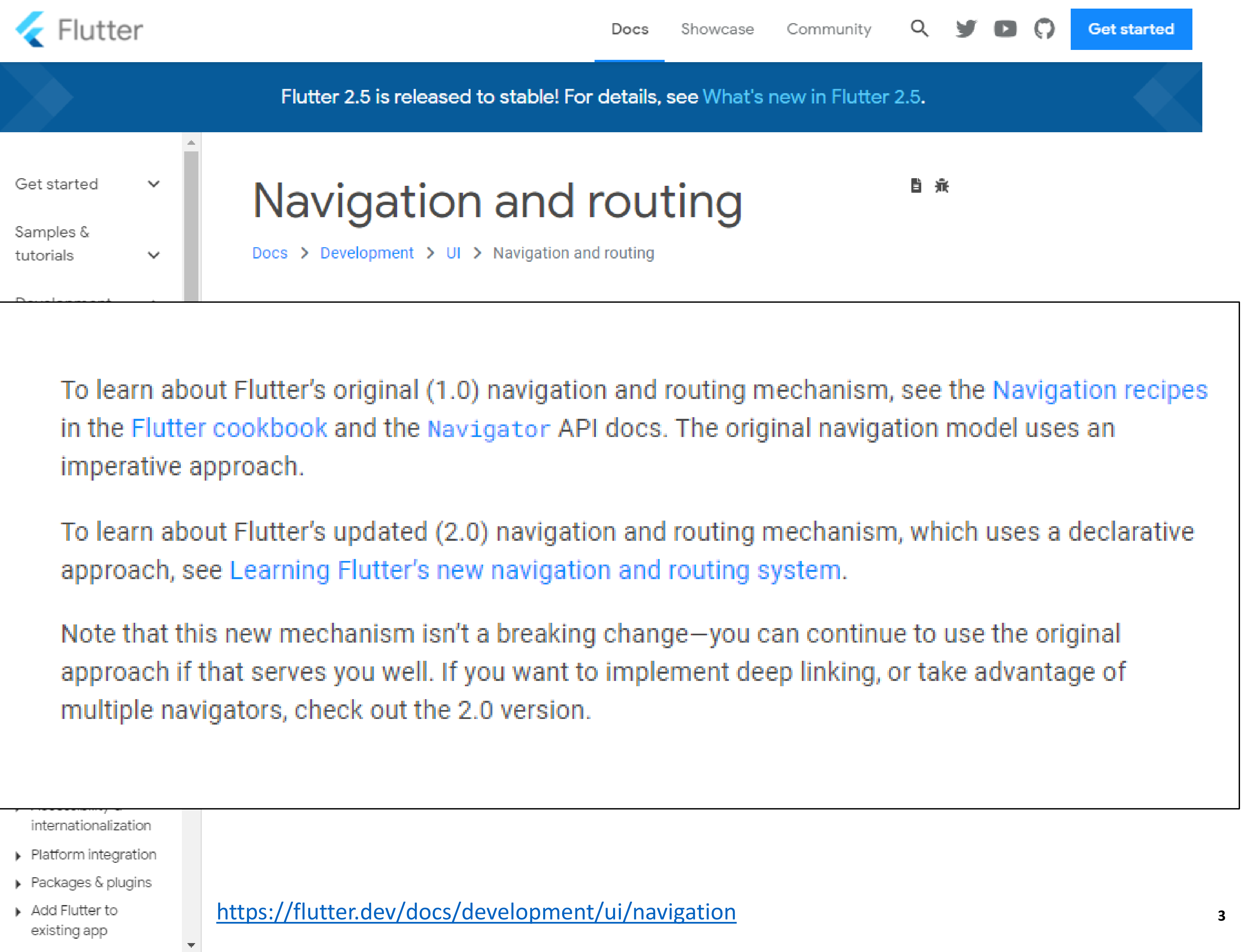
Docs > Development > UI > Navigation and routing

To learn about Flutter's original (1.0) navigation and routing mechanism, see the Navigation recipes in the Flutter cookbook and the `Navigator` API docs. The original navigation model uses an imperative approach.

To learn about Flutter's updated (2.0) navigation and routing mechanism, which uses a declarative approach, see Learning Flutter's new navigation and routing system.

Note that this new mechanism isn't a breaking change—you can continue to use the original approach if that serves you well. If you want to implement deep linking, or take advantage of multiple navigators, check out the 2.0 version.

https://flutter.dev/docs/development/ui/navigation



Flutter 2.5 is released to stable! For details, see What's new in Flutter 2.5.

Navigation and routing

Docs > Development > UI > Navigation and routing

To learn about Flutter's original (1.0) navigation and routing mechanism, see the Navigation recipes in the Flutter cookbook and the `Navigator` API docs. The original navigation model uses an imperative approach.

To learn about Flutter's updated (2.0) navigation and routing mechanism, which uses a declarative approach, see Learning Flutter's new navigation and routing system.

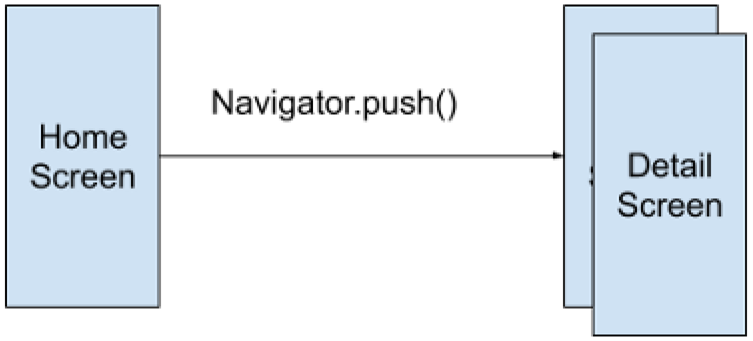
Note that this new mechanism isn't a breaking change—you can continue to use the original approach if that serves you well. If you want to implement deep linking, or take advantage of multiple navigators, check out the 2.0 version.

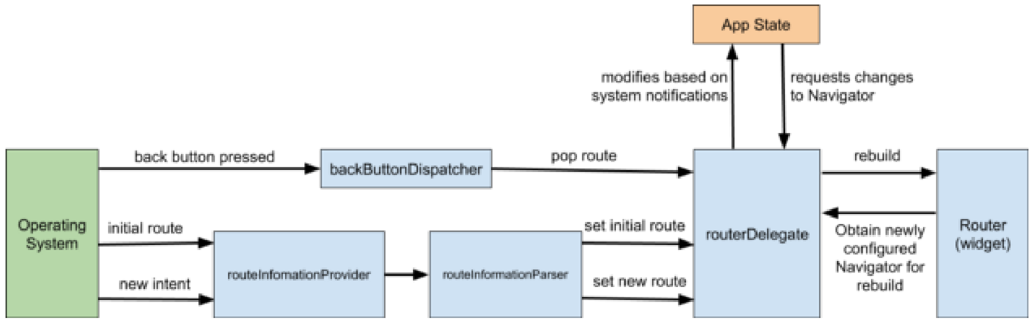
https://flutter.dev/docs/development/ui/navigation

Navigator 1.0 vs Navigator 2.0

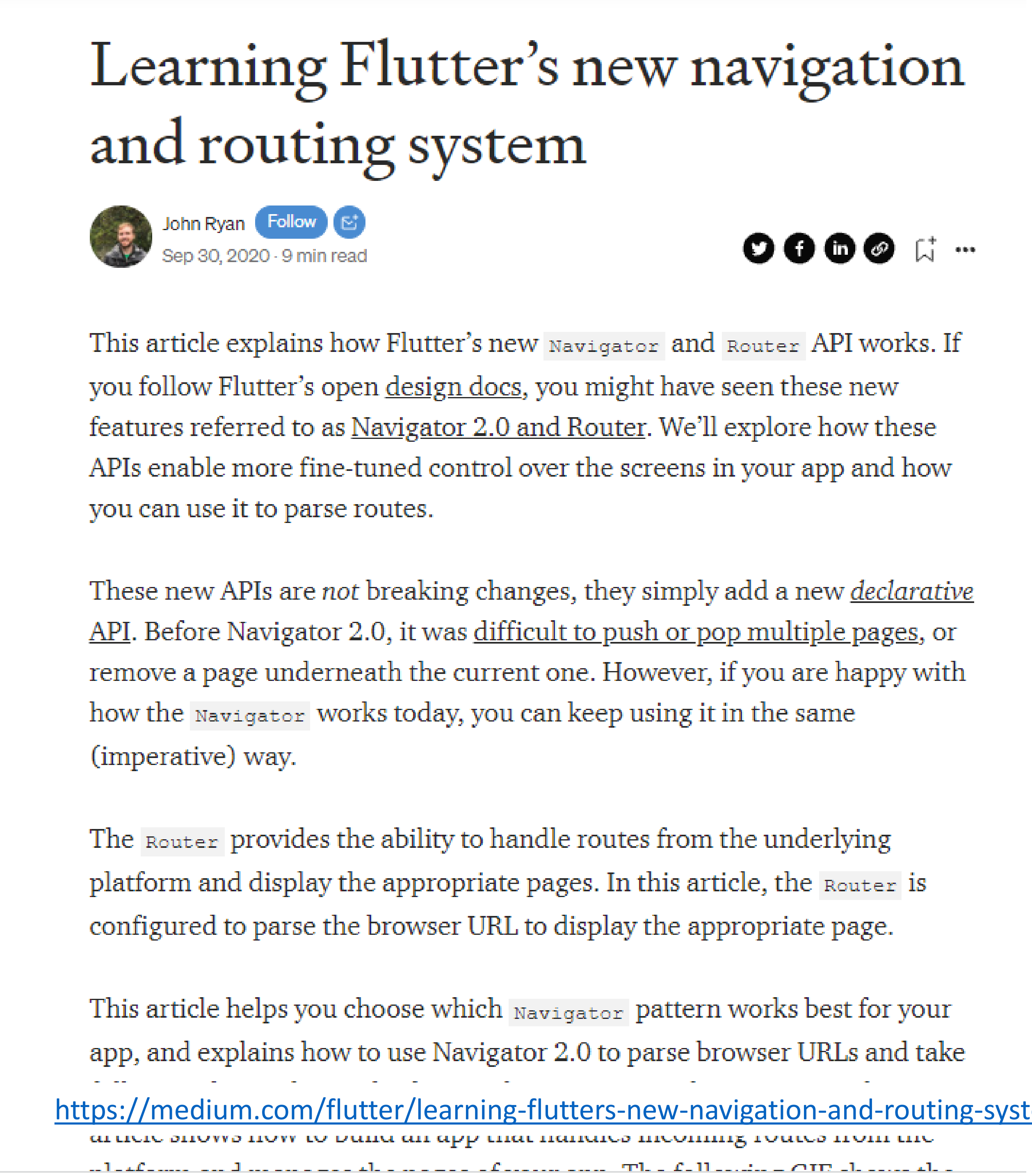
- imperative: Go this way
- Navigator
  - a widget that manages a stack of Route objects.
  - Manage the widgets / screen
  - Behaviour as stack
- Route:
  - The rules

- Declarative: context/state + current view define next route
- RouterDelegate
  - defines app-specific behavior of how the Router learns about changes in app state and how it responds to them.
- Split between programming and layout





universidade de aveiro



Learning Flutter's new navigation and routing system

John Ryan Follow

Sep 30, 2020 · 9 min read

This article explains how Flutter's new `Navigator` and `Router` API works. If you follow Flutter's open design docs, you might have seen these new features referred to as Navigator 2.0 and Router. We'll explore how these APIs enable more fine-tuned control over the screens in your app and how you can use it to parse routes.

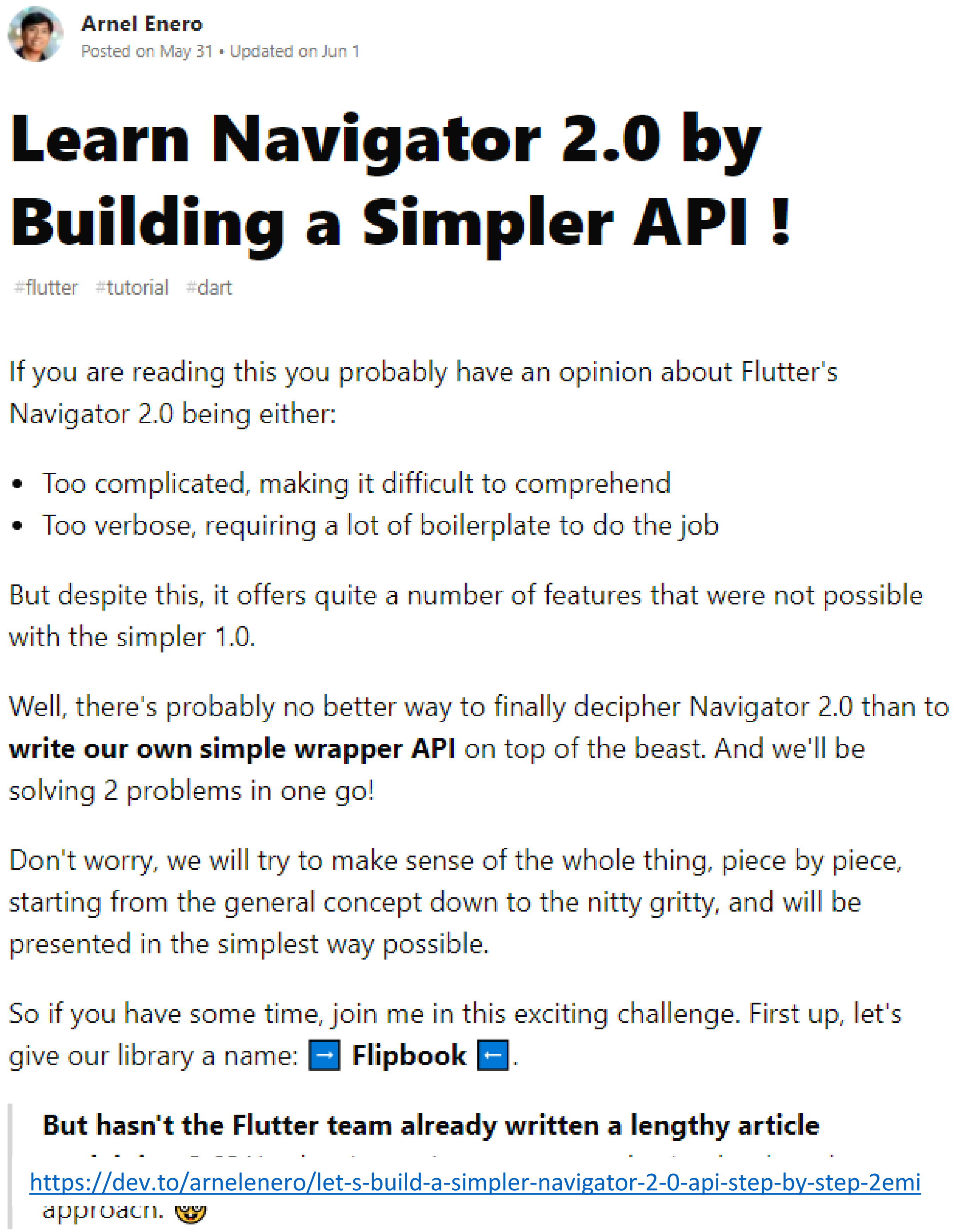
These new APIs are *not* breaking changes, they simply add a new *declarative* API. Before Navigator 2.0, it was difficult to push or pop multiple pages, or remove a page underneath the current one. However, if you are happy with how the `Navigator` works today, you can keep using it in the same (imperative) way.

The `Router` provides the ability to handle routes from the underlying platform and display the appropriate pages. In this article, the `Router` is configured to parse the browser URL to display the appropriate page.

This article helps you choose which `Navigator` pattern works best for your app, and explains how to use Navigator 2.0 to parse browser URLs and take

article shows how to build an app that handles incoming routes from the

https://medium.com/flutter/learning-flutters-new-navigation-and-routing-system-7c9068155ade



Arnel Enero
Posted on May 31 • Updated on Jun 1

Learn Navigator 2.0 by Building a Simpler API !

#flutter #tutorial #dart

If you are reading this you probably have an opinion about Flutter's Navigator 2.0 being either:

- Too complicated, making it difficult to comprehend
- Too verbose, requiring a lot of boilerplate to do the job

But despite this, it offers quite a number of features that were not possible with the simpler 1.0.

Well, there's probably no better way to finally decipher Navigator 2.0 than to **write our own simple wrapper API** on top of the beast. And we'll be solving 2 problems in one go!

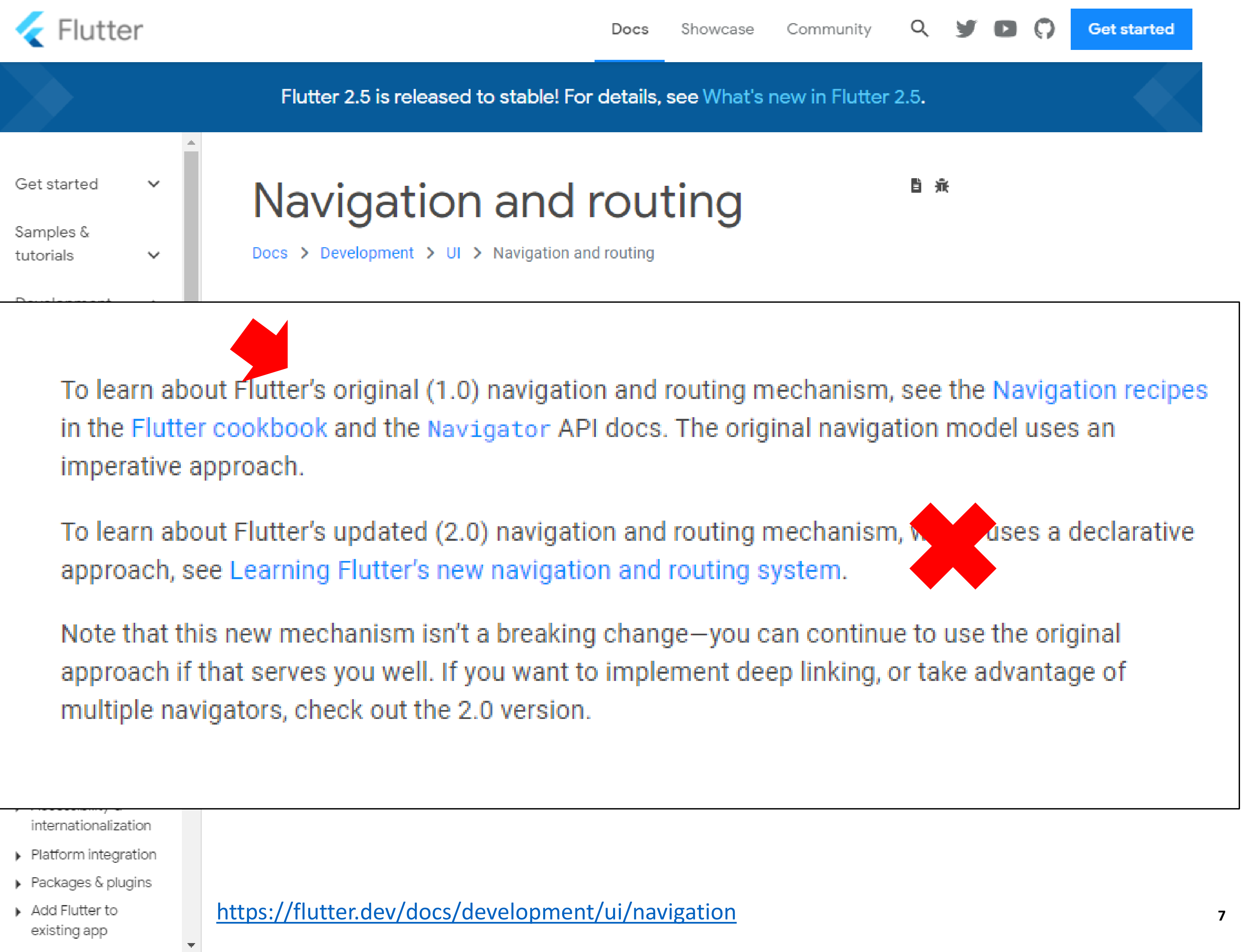
Don't worry, we will try to make sense of the whole thing, piece by piece, starting from the general concept down to the nitty gritty, and will be presented in the simplest way possible.

So if you have some time, join me in this exciting challenge. First up, let's give our library a name: ➡ **Flipbook** ⬅.

> **But hasn't the Flutter team already written a lengthy article**
>
> approach. 😀

https://dev.to/arnelenero/let-s-build-a-simpler-navigator-2-0-api-step-by-step-2emi

universidade de aveiro



Flutter Docs Showcase Community Get started

Flutter 2.5 is released to stable! For details, see What's new in Flutter 2.5.

Get started

Samples & tutorials

Navigation and routing

Docs > Development > UI > Navigation and routing

To learn about Flutter's original (1.0) navigation and routing mechanism, see the Navigation recipes in the Flutter cookbook and the `Navigator` API docs. The original navigation model uses an imperative approach.

To learn about Flutter's updated (2.0) navigation and routing mechanism, which uses a declarative approach, see Learning Flutter's new navigation and routing system.

Note that this new mechanism isn't a breaking change—you can continue to use the original approach if that serves you well. If you want to implement deep linking, or take advantage of multiple navigators, check out the 2.0 version.

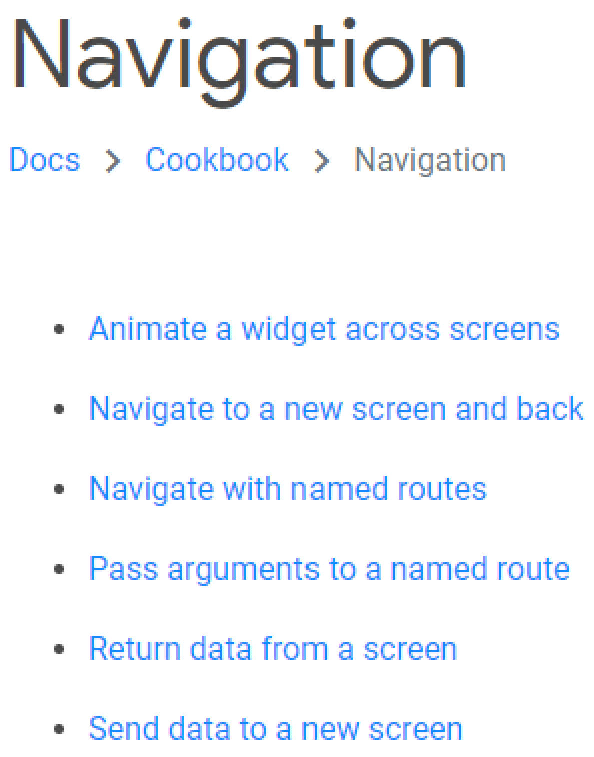
internationalization

Platform integration

Packages & plugins

Add Flutter to existing app

https://flutter.dev/docs/development/ui/navigation



Navigation

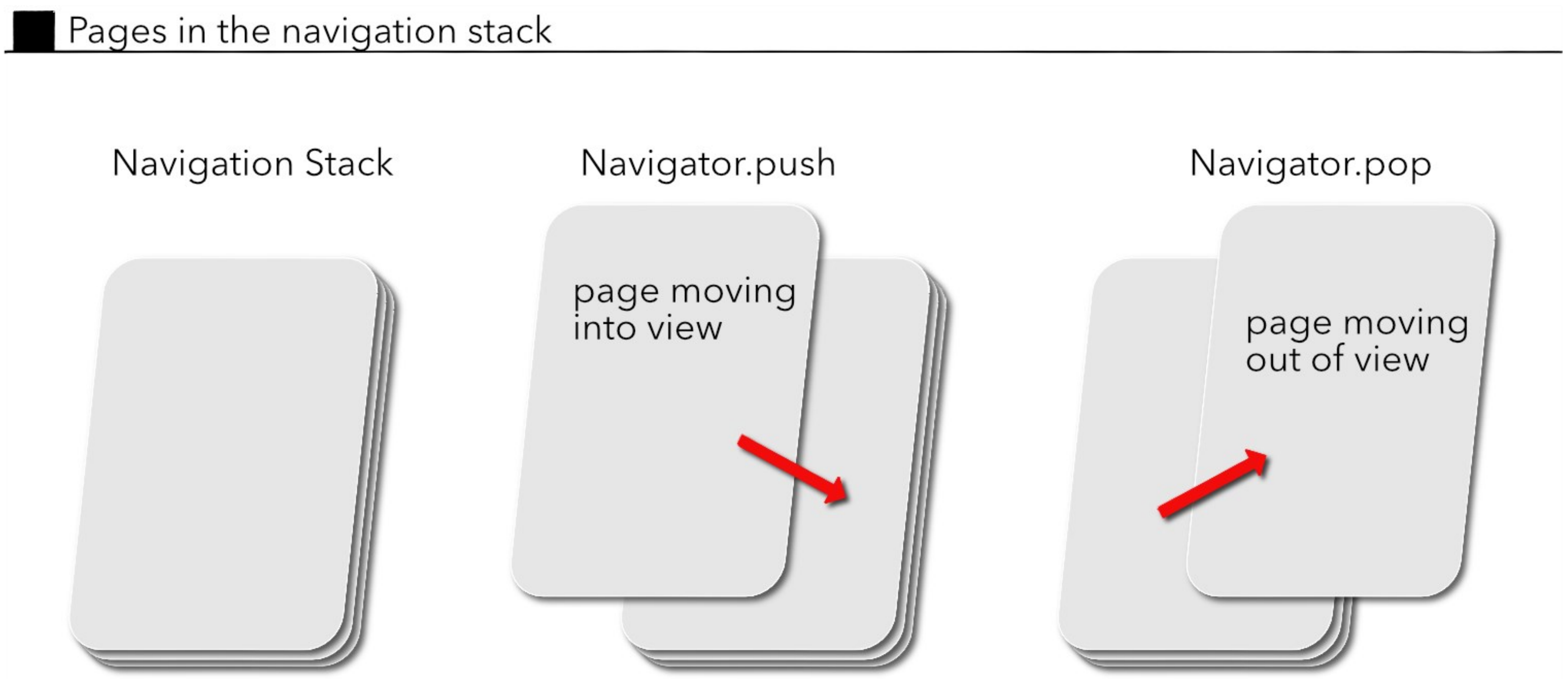
Docs > Cookbook > Navigation

- Animate a widget across screens
- Navigate to a new screen and back
- Navigate with named routes
- Pass arguments to a named route
- Return data from a screen
- Send data to a new screen

https://flutter.dev/docs/cookbook/navigation

Navigator 1.0

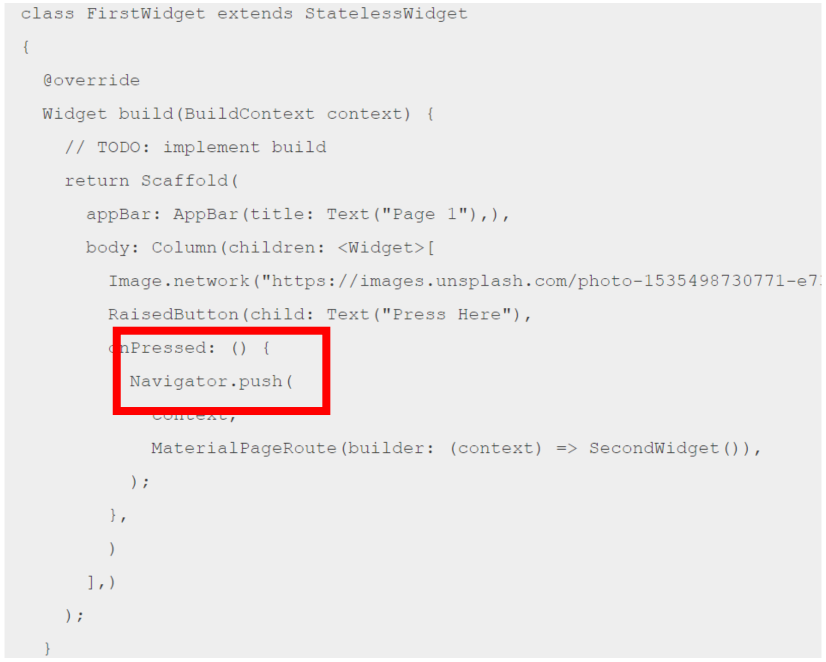
Navigator “singleton” manages views stack



https://androidmonks.com/routing-navigation-flutter/

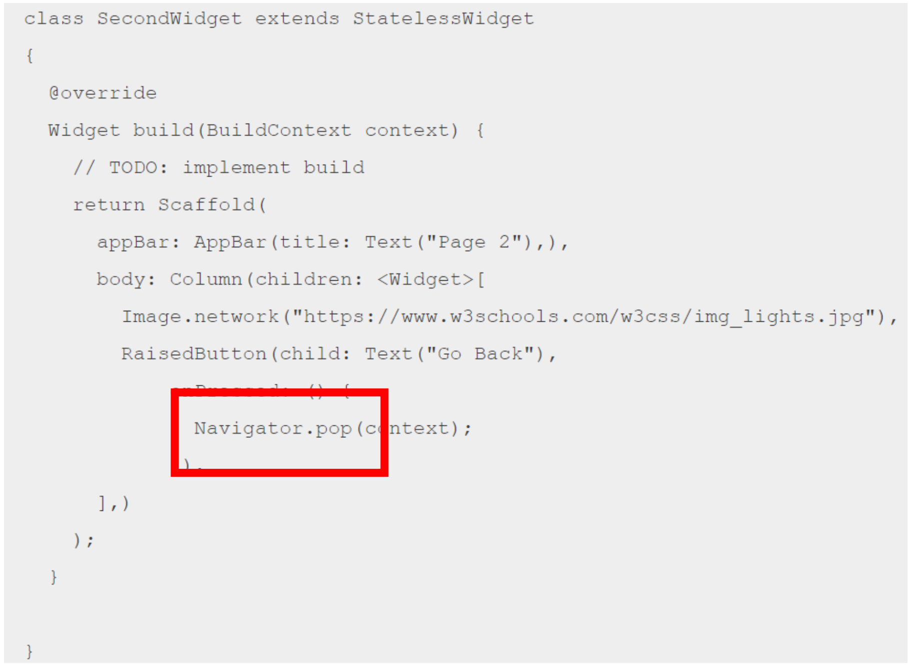
Navigation and screens

- Based on a controller: Navigator
- Responsible for managing the screen that is on top



```
class FirstWidget extends StatelessWidget
{
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return Scaffold(
      appBar: AppBar(title: Text("Page 1"),),
      body: Column(children: <Widget>[
        Image.network("https://images.unsplash.com/photo-1535498730771-e7
        RaisedButton(child: Text("Press Here"),
        onPressed: () {
          Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => SecondWidget()),
          );
        },
        )
      ],)
    );
  }
```

It push images to the “screen”

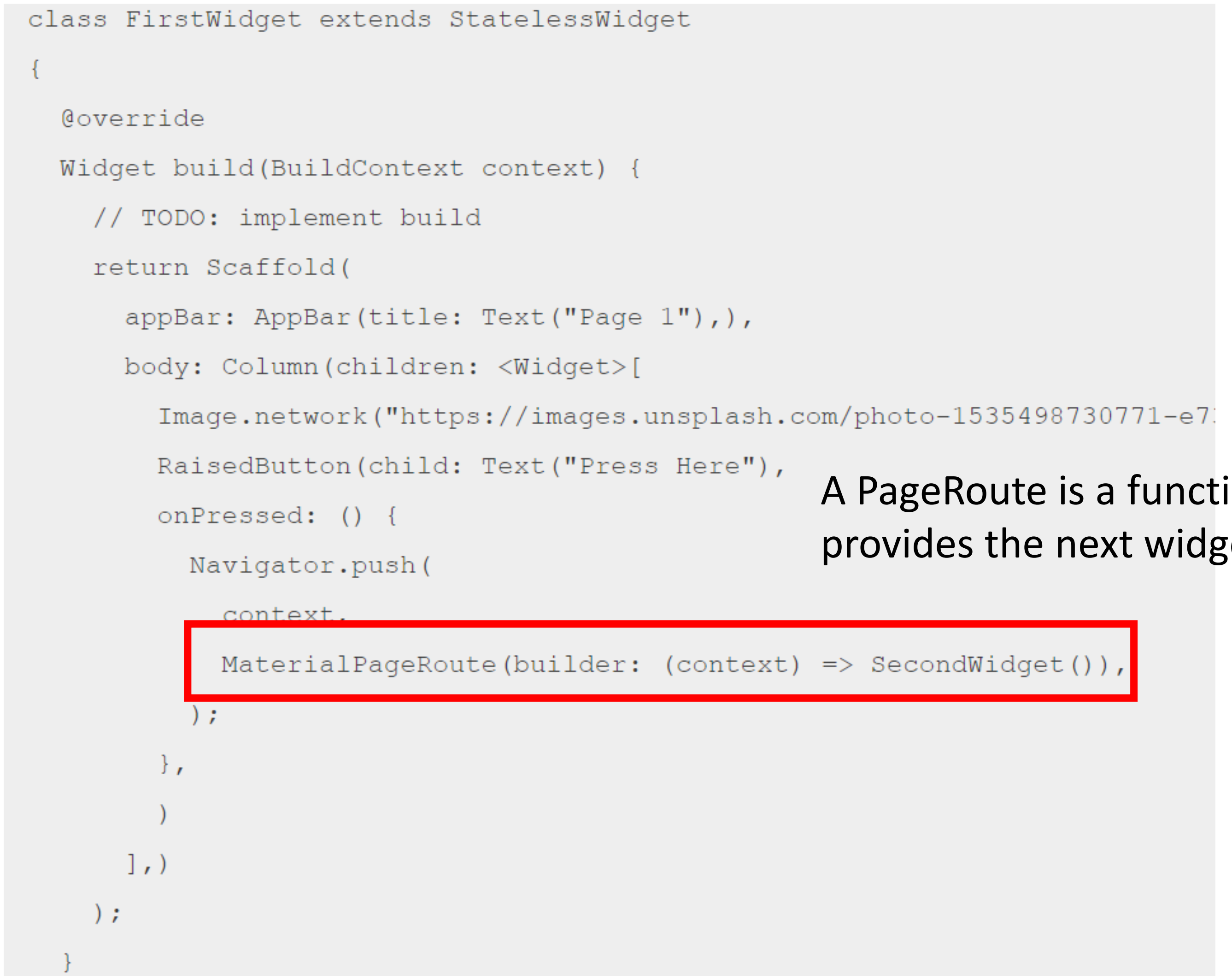


```
class SecondWidget extends StatelessWidget
{
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return Scaffold(
      appBar: AppBar(title: Text("Page 2"),),
      body: Column(children: <Widget>[
        Image.network("https://www.w3schools.com/w3css/img_lights.jpg"),
        RaisedButton(child: Text("Go Back"),
              onPressed: () {
                Navigator.pop(context);
              },
      ],)
    );
  }

}
```

To dismiss screen they must be popped

universidade de aveiro

When pushing it must provide a route to the next widget



```
class FirstWidget extends StatelessWidget
{
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return Scaffold(
      appBar: AppBar(title: Text("Page 1"),),
      body: Column(children: <Widget>[
        Image.network("https://images.unsplash.com/photo-1535498730771-e7
        RaisedButton(child: Text("Press Here"),
        onPressed: () {
          Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => SecondWidget()),
          );
        },
        )
      ],)
    );
  }
```

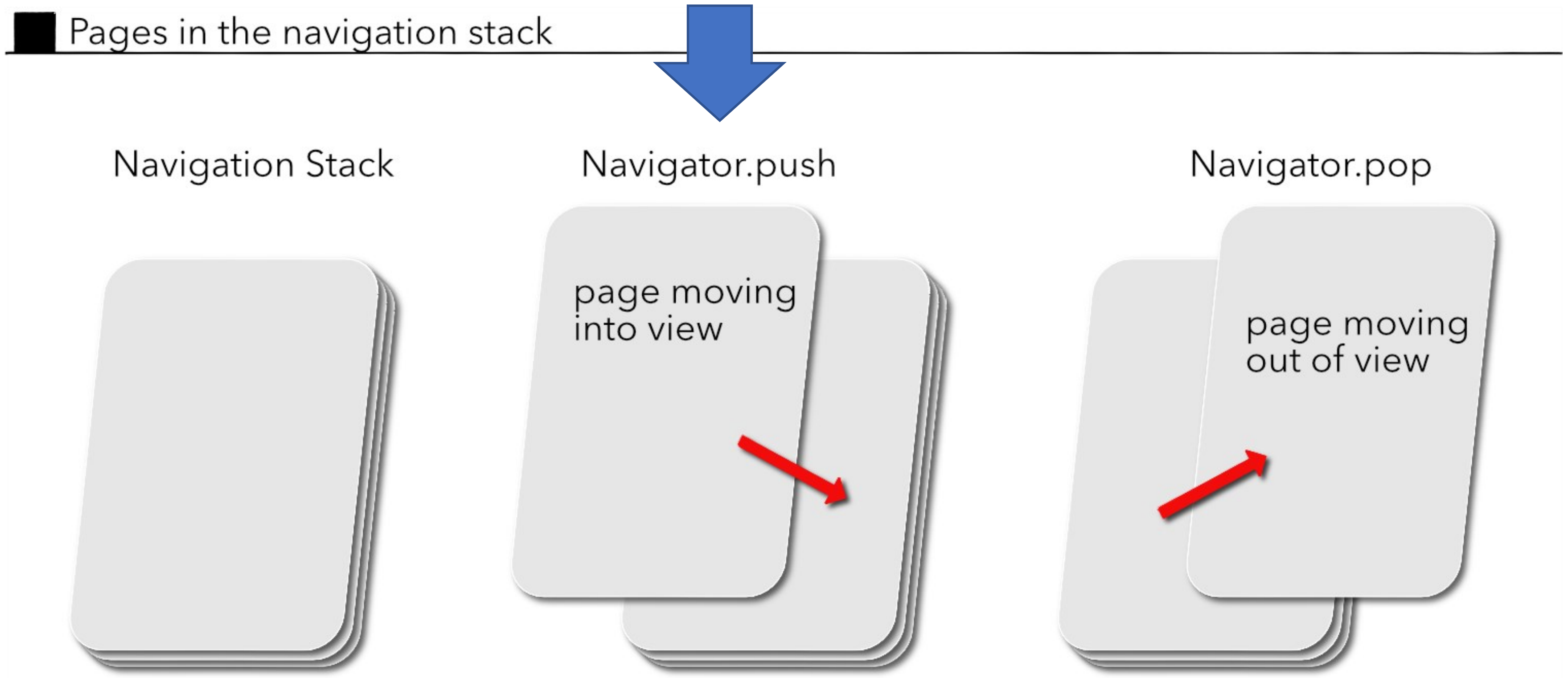
A PageRoute is a function that from context provides the next widget (view to show)

universidade de aveiro

Not needed when popping – just leave the top…

```
class SecondWidget extends StatelessWidget
{
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return Scaffold(
      appBar: AppBar(title: Text("Page 2"),),
      body: Column(children: <Widget>[
        Image.network("https://www.w3schools.com/w3css/img_lights.jpg"),
        RaisedButton(child: Text("Go Back"),
            onPressed: () {
              Navigator.pop(context);
            }),
      ],)
    );
  }

}
```

Changing view i.e. pushing widget into stack



https://androidmonks.com/routing-navigation-flutter/

Going to a view: push the stack

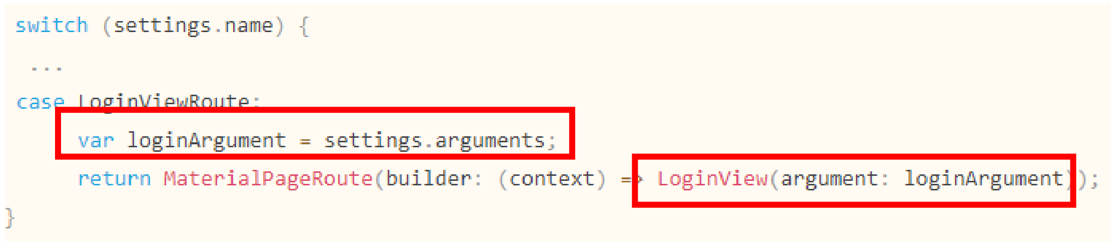
```
 // Perform navigation to LoginView
Navigator.pushNamed(context, LoginViewRoute);
```

Pass as an extra argument as a plain object

```
Navigator.pushNamed(context, LoginViewRoute, arguments: 'Data Passed in');
```

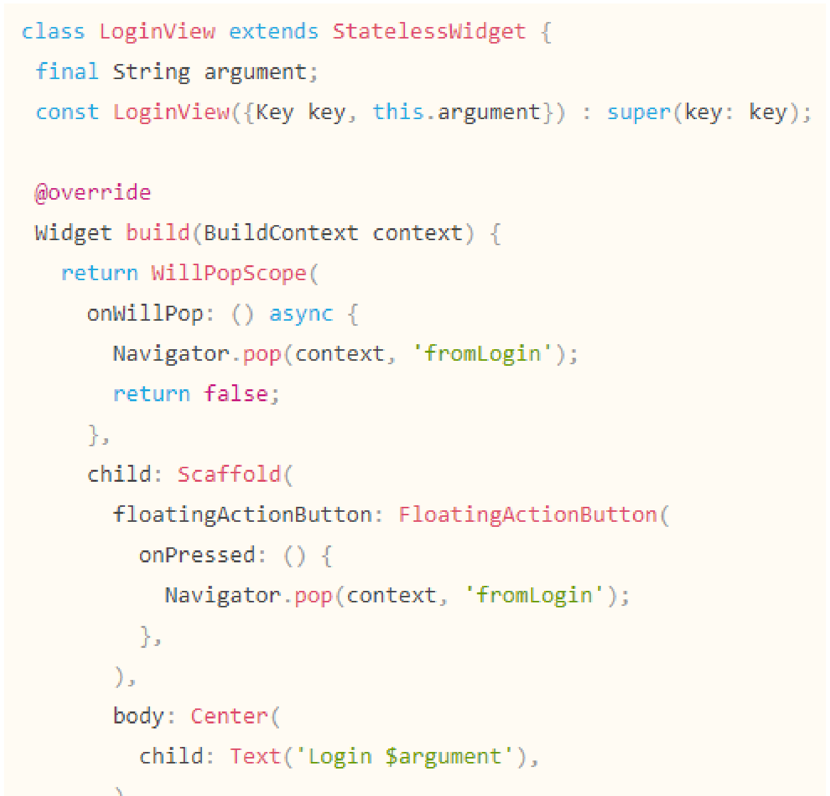
Get the passed argument

Get argument in routing



```
switch (settings.name) {
 ...
case LoginViewRoute:
     var loginArgument = settings.arguments;
     return MaterialPageRoute(builder: (context) => LoginView(argument: loginArgument));
}
```

Must provide constructor



```
class LoginView extends StatelessWidget {
 final String argument;
 const LoginView({Key key, this.argument}) : super(key: key);

 @override
 Widget build(BuildContext context) {
   return WillPopScope(
     onWillPop: () async {
       Navigator.pop(context, 'fromLogin');
       return false;
     },
     child: Scaffold(
       floatingActionButton: FloatingActionButton(
         onPressed: () {
           Navigator.pop(context, 'fromLogin');
         },
       ),
       body: Center(
         child: Text('Login $argument'),
```

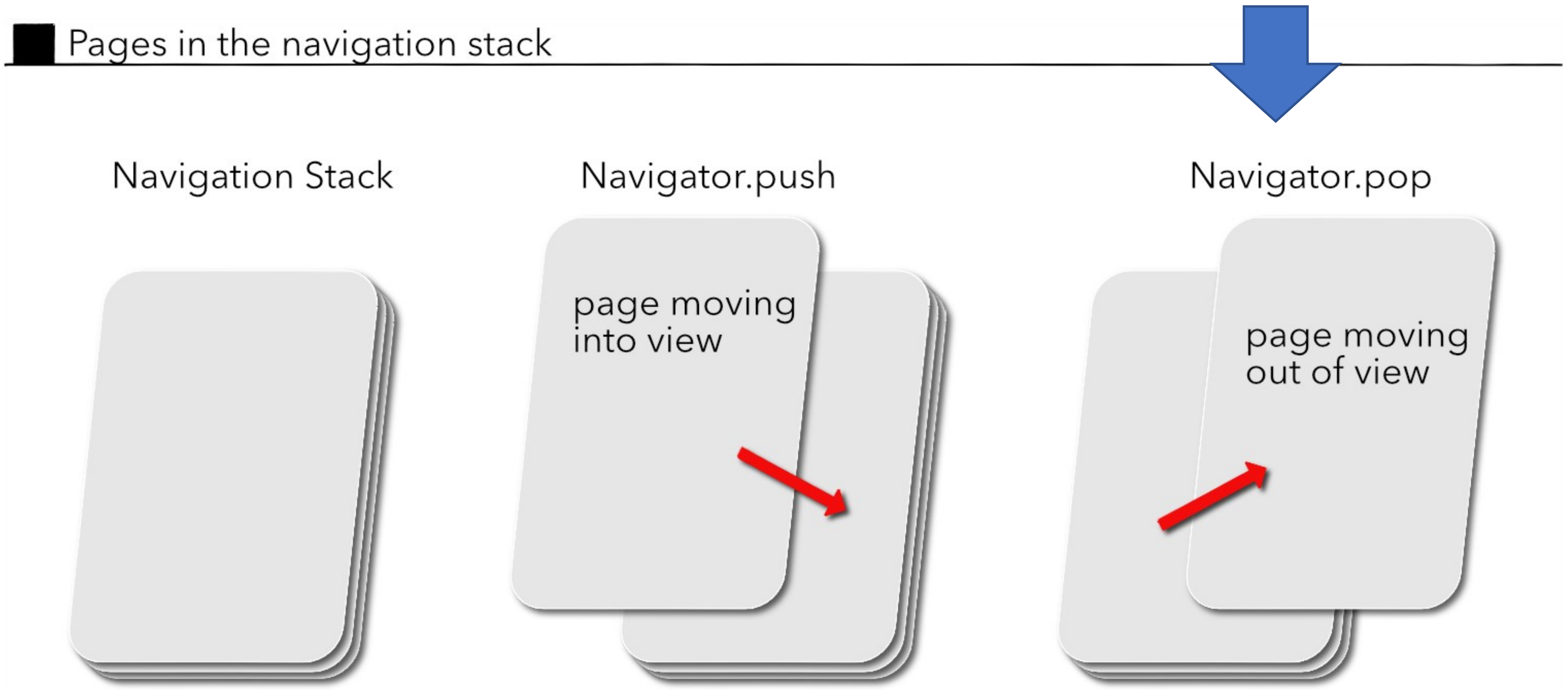
Pass it … this case via constructor

Get argument in routing

```
switch (settings.name) {
 ...
case LoginViewRoute:
     var loginArgument = settings.arguments;
     return MaterialPageRoute(builder: (context) => LoginView(argument: loginArgument));
}
```

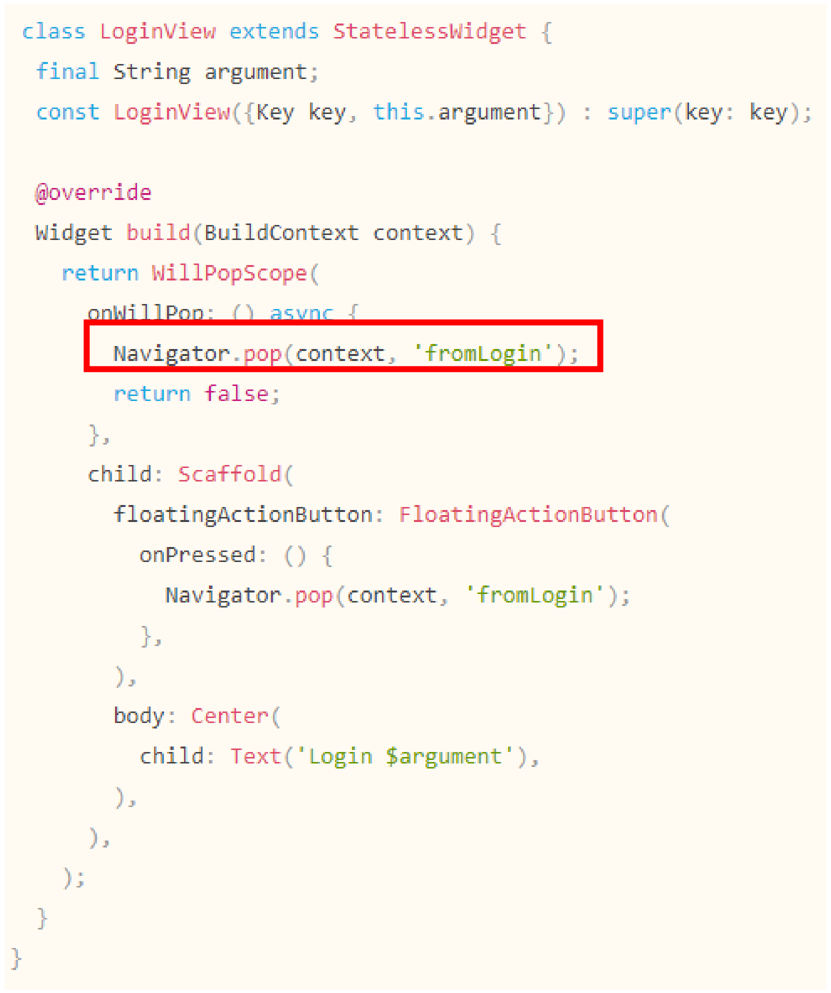
```
class LoginView extends StatelessWidget {
 final String argument;
 const LoginView({Key key, this.argument}) : super(key: key);

 @override
 Widget build(BuildContext context) {
   return WillPopScope(
     onWillPop: () async {
       Navigator.pop(context, 'fromLogin');
       return false;
     },
     child: Scaffold(
       floatingActionButton: FloatingActionButton(
         onPressed: () {
           Navigator.pop(context, 'fromLogin');
         },
       ),
       body: Center(
         child: Text('Login $argument'),
```

Returning to preview widget i.e. pop stack



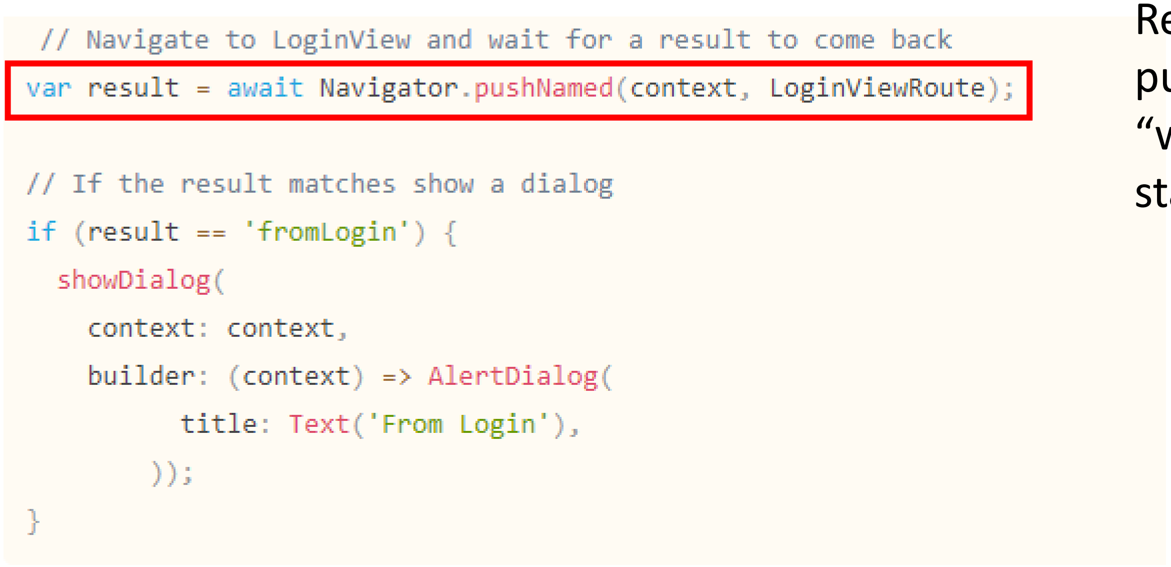
https://androidmonks.com/routing-navigation-flutter/

Return to previous i.e. pop the view from stack



```
class LoginView extends StatelessWidget {
  final String argument;
  const LoginView({Key key, this.argument}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return WillPopScope(
      onWillPop: () async {
        Navigator.pop(context, 'fromLogin');
        return false;
      },
      child: Scaffold(
        floatingActionButton: FloatingActionButton(
          onPressed: () {
            Navigator.pop(context, 'fromLogin');
          },
        ),
        body: Center(
          child: Text('Login $argument'),
        ),
      ),
    );
  }
}
```

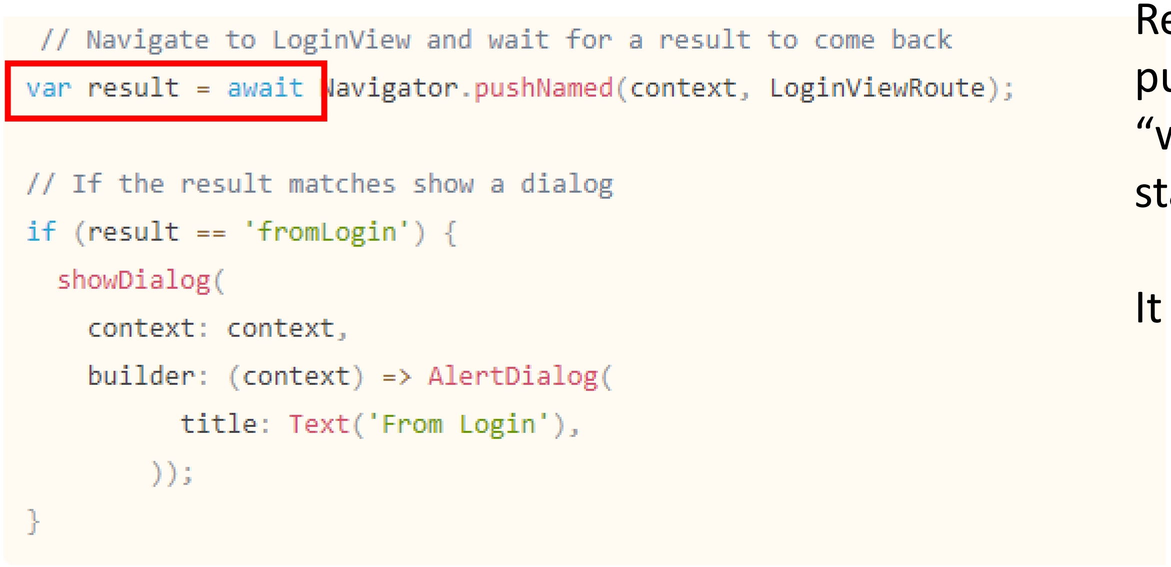
To get answer must wait for Future answer



```
 // Navigate to LoginView and wait for a result to come back
var result = await Navigator.pushNamed(context, LoginViewRoute);

// If the result matches show a dialog
if (result == 'fromLogin') {
  showDialog(
    context: context,
    builder: (context) => AlertDialog(
          title: Text('From Login'),
        ));
}
```

Remember that the view pushing the view was “waiting” in the Navigator stack

To get answer must wait for Future answer

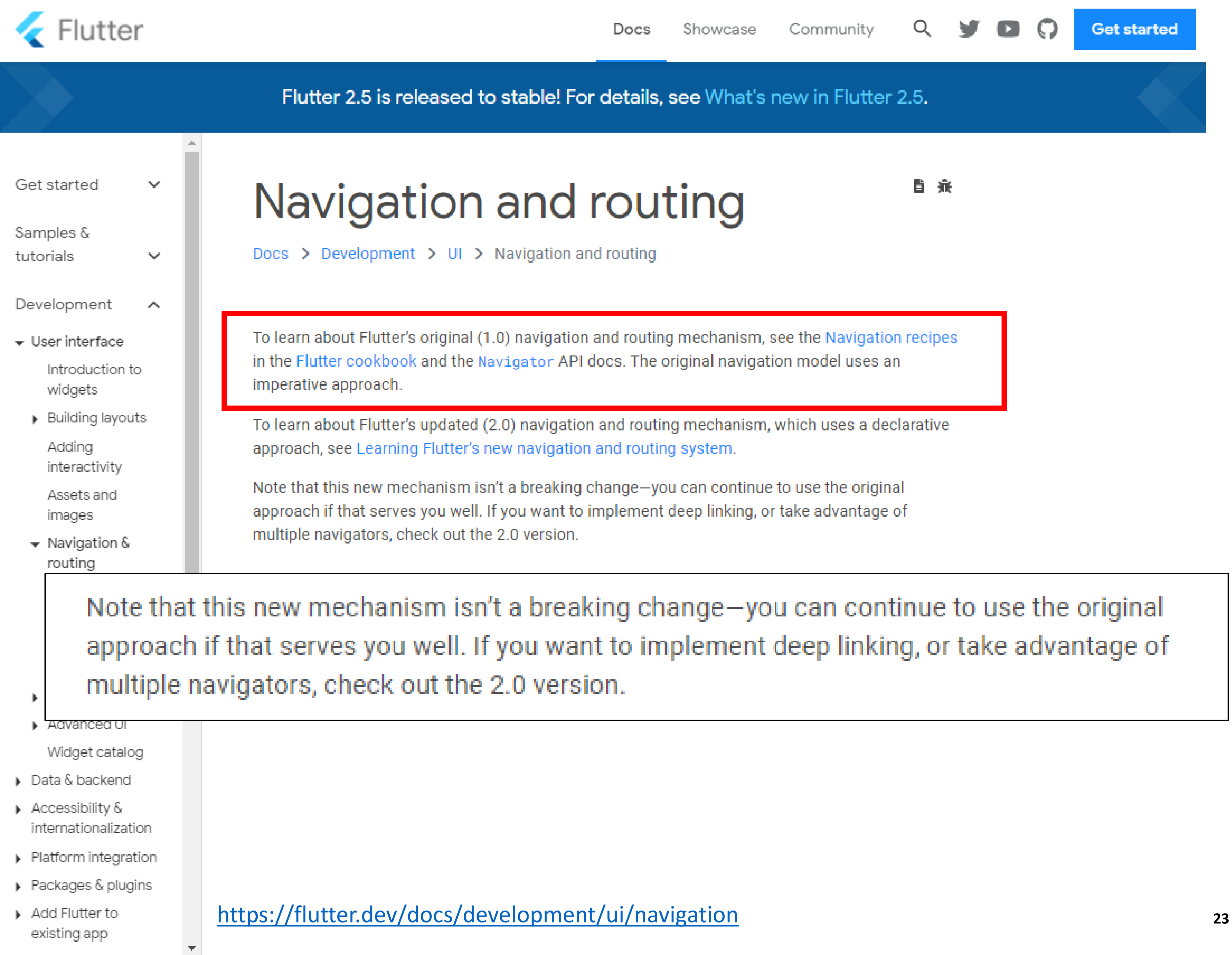


```
 // Navigate to LoginView and wait for a result to come back
var result = await Navigator.pushNamed(context, LoginViewRoute);

// If the result matches show a dialog
if (result == 'fromLogin') {
  showDialog(
    context: context,
    builder: (context) => AlertDialog(
          title: Text('From Login'),
        ));
}
```

Remember that the view pushing the view was "waiting" in the Navigator stack

It was waiting for a "pop"

Basic navigation examples



Flutter 2.5 is released to stable! For details, see What's new in Flutter 2.5.

Navigation and routing

Docs > Development > UI > Navigation and routing

To learn about Flutter's original (1.0) navigation and routing mechanism, see the Navigation recipes in the Flutter cookbook and the `Navigator` API docs. The original navigation model uses an imperative approach.

To learn about Flutter's updated (2.0) navigation and routing mechanism, which uses a declarative approach, see Learning Flutter's new navigation and routing system.

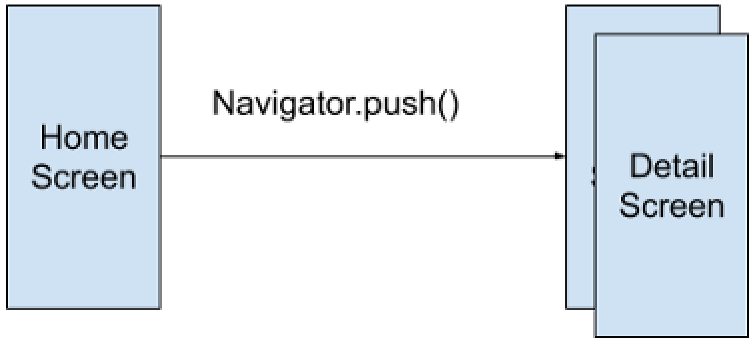
Note that this new mechanism isn't a breaking change—you can continue to use the original approach if that serves you well. If you want to implement deep linking, or take advantage of multiple navigators, check out the 2.0 version.

Note that this new mechanism isn't a breaking change—you can continue to use the original approach if that serves you well. If you want to implement deep linking, or take advantage of multiple navigators, check out the 2.0 version.

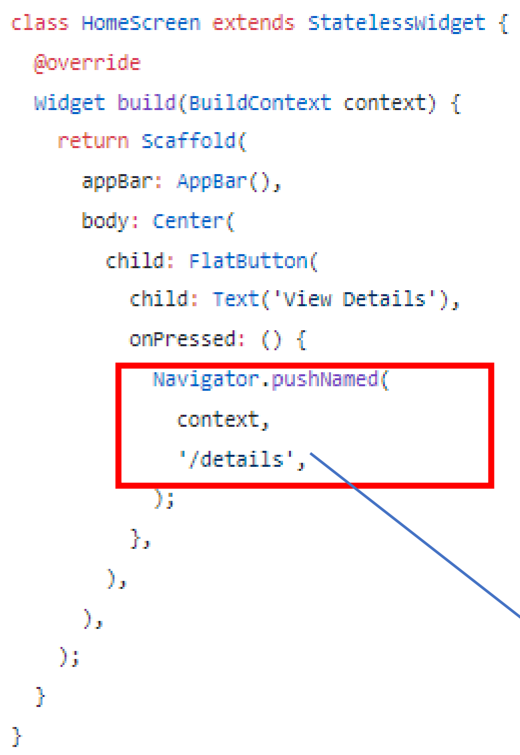
https://flutter.dev/docs/development/ui/navigation

Navigator 1.0: defining routes

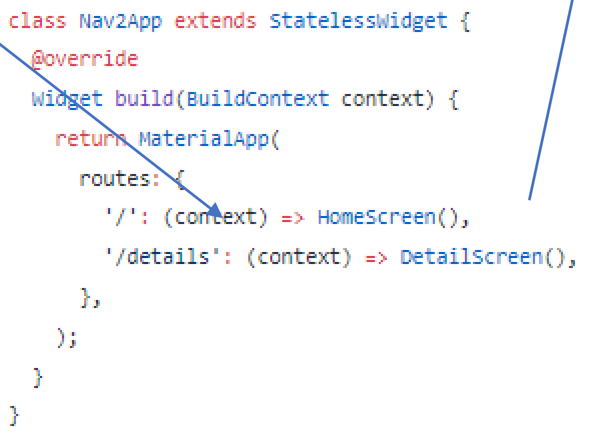
```
class Nav2App extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      routes: {
        '/': (context) => HomeScreen(),
        '/details': (context) => DetailScreen(),
      },
    );
  }
}
```



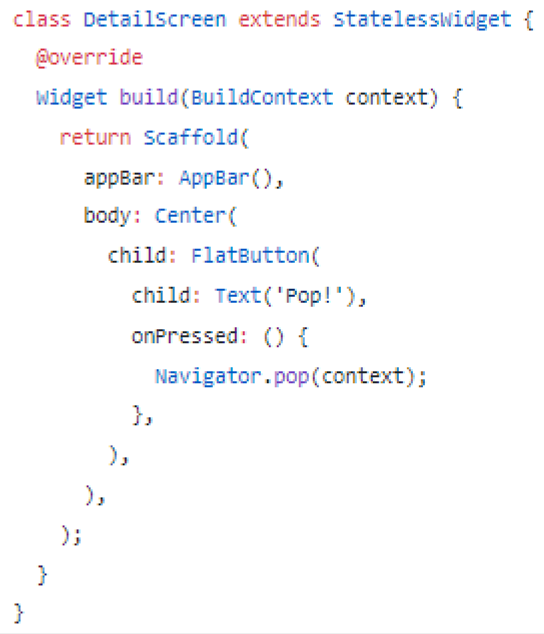
Navigator 1.0: navigating screens



```
class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(),
      body: Center(
        child: FlatButton(
          child: Text('View Details'),
          onPressed: () {
            Navigator.pushNamed(
              context,
              '/details',
            );
          },
        ),
      ),
    );
  }
}
```



```
class Nav2App extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      routes: {
        '/': (context) => HomeScreen(),
        '/details': (context) => DetailScreen(),
      },
    );
  }
}
```



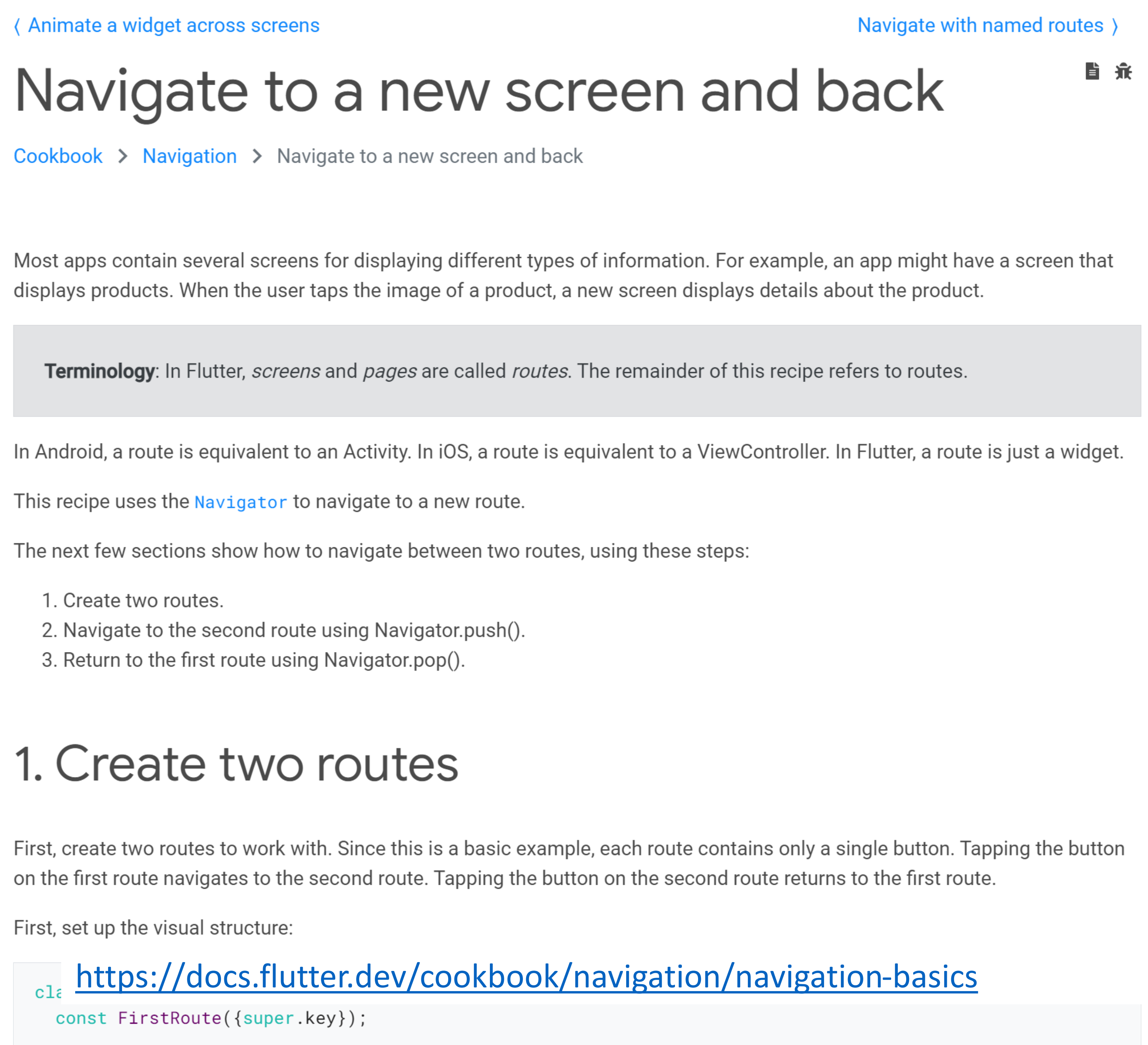
```
class DetailScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(),
      body: Center(
        child: FlatButton(
          child: Text('Pop!'),
          onPressed: () {
            Navigator.pop(context);
          },
        ),
      ),
    );
  }
}
```

Navigator 1.0: navigating screens

```
class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(),
      body: Center(
        child: FlatButton(
          child: Text('View Details'),
          onPressed: () {
            Navigator.pushNamed(
              context,
              '/details',
            );
          },
        ),
      ),
    );
  }
}
```

```
class DetailScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(),
      body: Center(
        child: FlatButton(
          child: Text('Pop!'),
          onPressed: () {
            Navigator.pop(context);
          },
        ),
      ),
    );
  }
}
```

```
class Nav2App extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      routes: {
        '/': (context) => HomeScreen(),
        '/details': (context) => DetailScreen(),
      },
    );
  }
}
```



⟨ Animate a widget across screens   Navigate with named routes ⟩

Navigate to a new screen and back

Cookbook > Navigation > Navigate to a new screen and back

Most apps contain several screens for displaying different types of information. For example, an app might have a screen that displays products. When the user taps the image of a product, a new screen displays details about the product.

Terminology: In Flutter, *screens* and *pages* are called *routes*. The remainder of this recipe refers to routes.

In Android, a route is equivalent to an Activity. In iOS, a route is equivalent to a ViewController. In Flutter, a route is just a widget.

This recipe uses the `Navigator` to navigate to a new route.

The next few sections show how to navigate between two routes, using these steps:

1. Create two routes.
2. Navigate to the second route using Navigator.push().
3. Return to the first route using Navigator.pop().

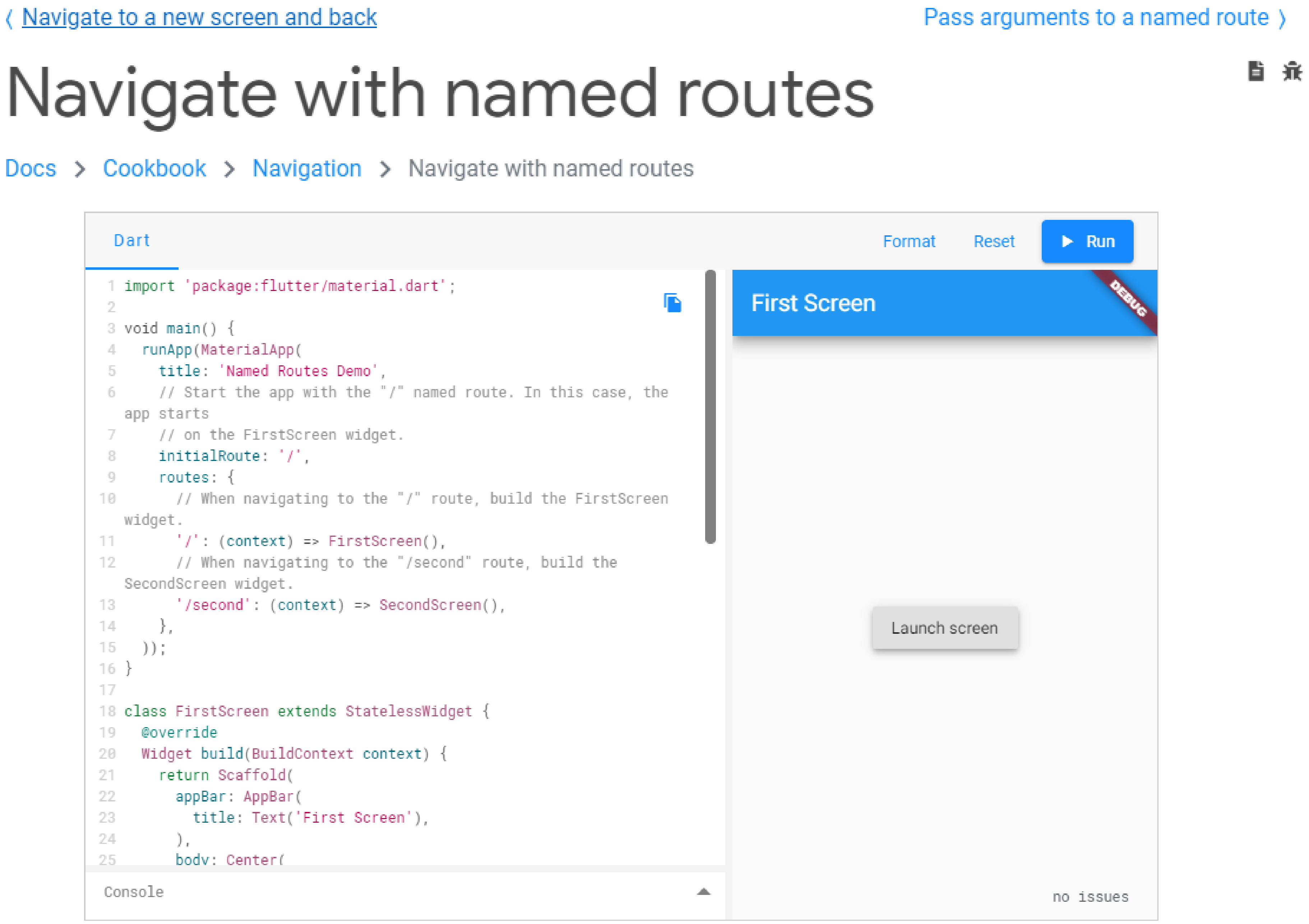
1. Create two routes

First, create two routes to work with. Since this is a basic example, each route contains only a single button. Tapping the button on the first route navigates to the second route. Tapping the button on the second route returns to the first route.

First, set up the visual structure:

```
cla
  const FirstRoute({super.key});
```

https://docs.flutter.dev/cookbook/navigation/navigation-basics



⟨ Navigate to a new screen and back　　　　Pass arguments to a named route ⟩

Navigate with named routes

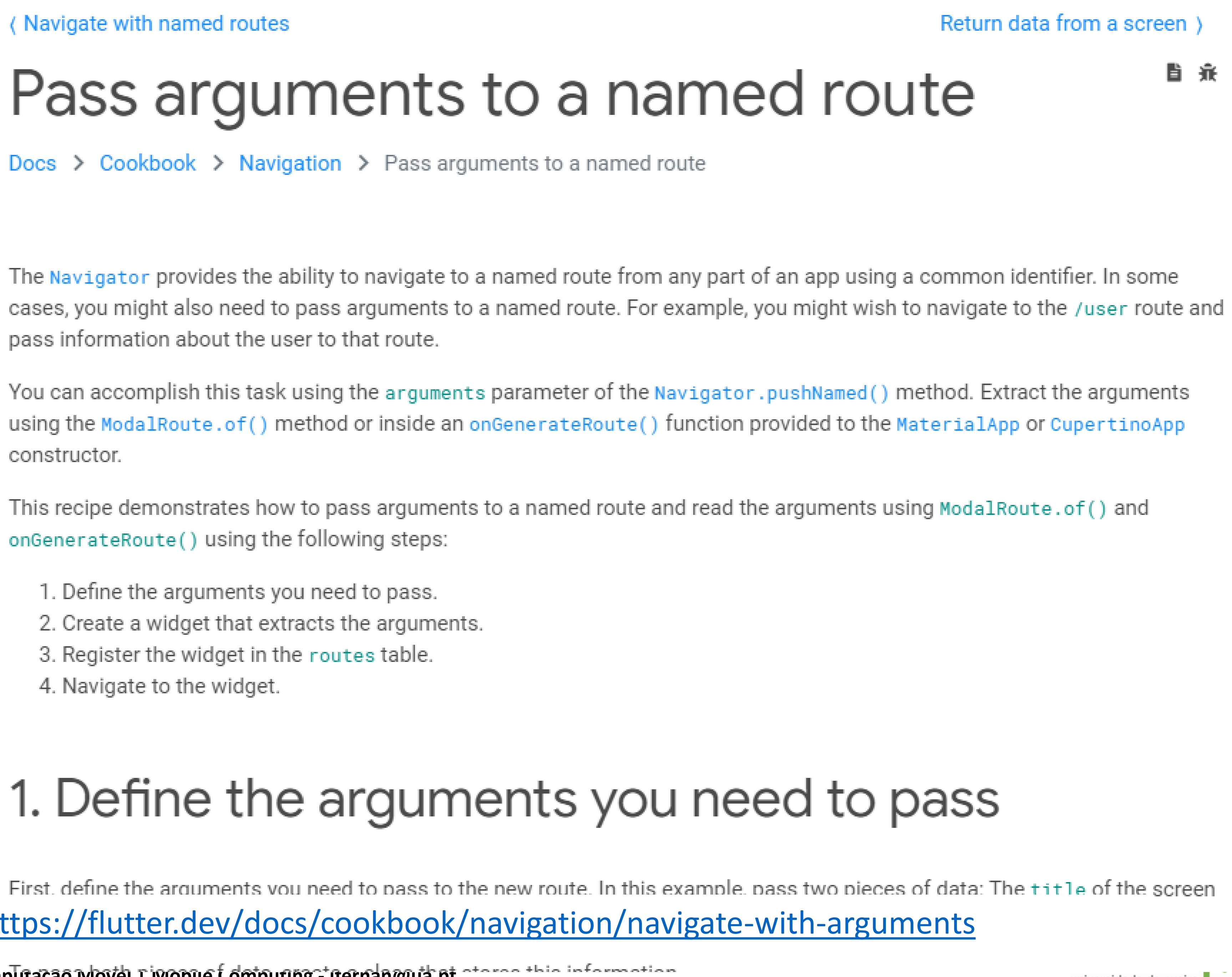
Docs > Cookbook > Navigation > Navigate with named routes

Dart　　Format　Reset　▶ Run

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

void main() {
  runApp(MaterialApp(
    title: 'Named Routes Demo',
    // Start the app with the "/" named route. In this case, the app starts
    // on the FirstScreen widget.
    initialRoute: '/',
    routes: {
      // When navigating to the "/" route, build the FirstScreen widget.
      '/': (context) => FirstScreen(),
      // When navigating to the "/second" route, build the SecondScreen widget.
      '/second': (context) => SecondScreen(),
    },
  ));
}

class FirstScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('First Screen'),
      ),
      body: Center(
```

Console

https://flutter.dev/docs/cookbook/navigation/named-routes



⟨ Navigate with named routes   Return data from a screen ⟩

Pass arguments to a named route

Docs > Cookbook > Navigation > Pass arguments to a named route

The `Navigator` provides the ability to navigate to a named route from any part of an app using a common identifier. In some cases, you might also need to pass arguments to a named route. For example, you might wish to navigate to the `/user` route and pass information about the user to that route.

You can accomplish this task using the `arguments` parameter of the `Navigator.pushNamed()` method. Extract the arguments using the `ModalRoute.of()` method or inside an `onGenerateRoute()` function provided to the `MaterialApp` or `CupertinoApp` constructor.

This recipe demonstrates how to pass arguments to a named route and read the arguments using `ModalRoute.of()` and `onGenerateRoute()` using the following steps:

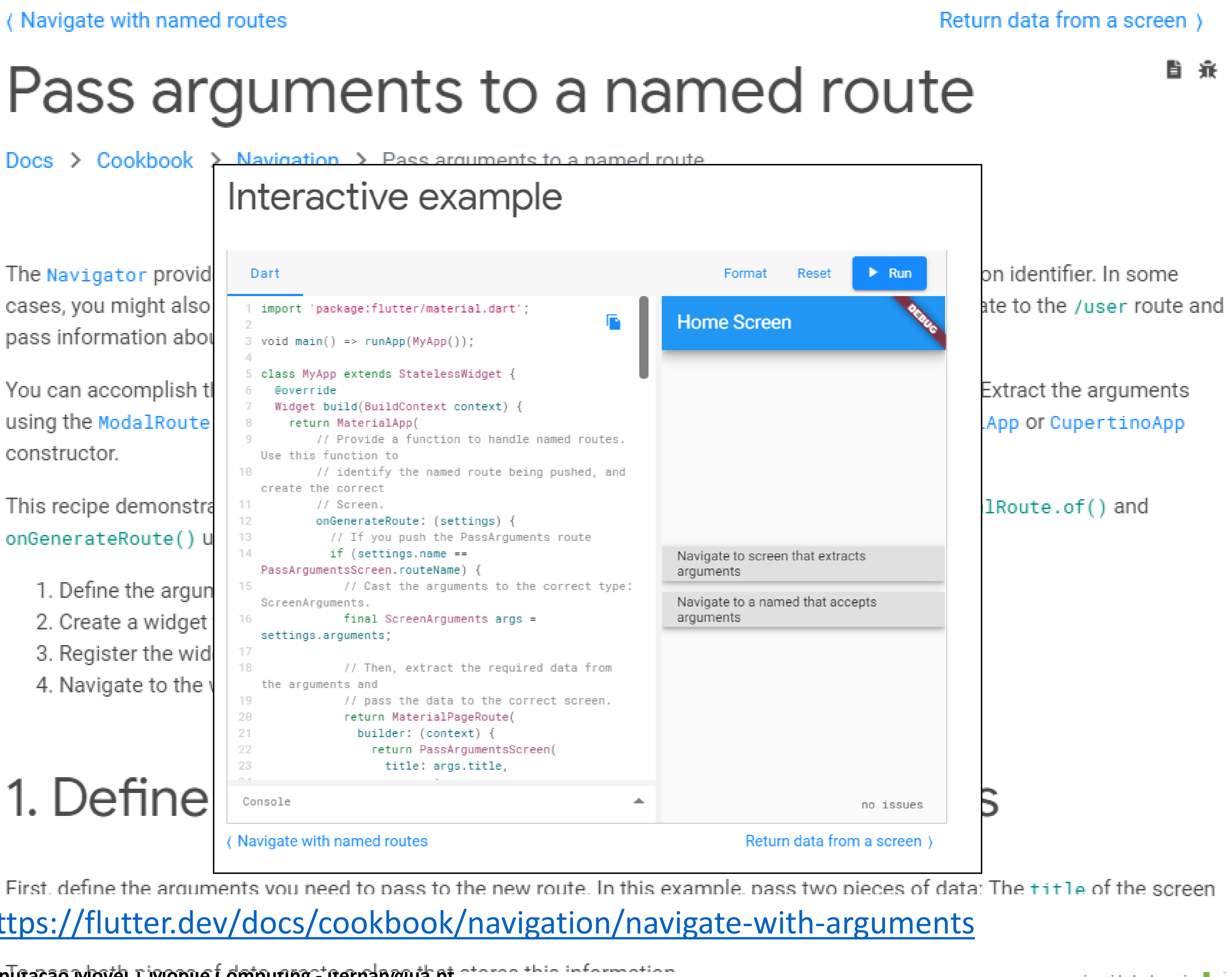
1. Define the arguments you need to pass.
2. Create a widget that extracts the arguments.
3. Register the widget in the `routes` table.
4. Navigate to the widget.

1. Define the arguments you need to pass

First, define the arguments you need to pass to the new route. In this example, pass two pieces of data: The `title` of the screen

https://flutter.dev/docs/cookbook/navigation/navigate-with-arguments

To pass both pieces of data, create a class that stores this information.



⟨ Navigate with named routes   Return data from a screen ⟩

Pass arguments to a named route

Docs › Cookbook › Navigation › Pass arguments to a named route

The Navigator provid… …on identifier. In some cases, you might also… …ate to the /user route and pass information abou…

You can accomplish t… …Extract the arguments using the ModalRoute… …App or CupertinoApp constructor.

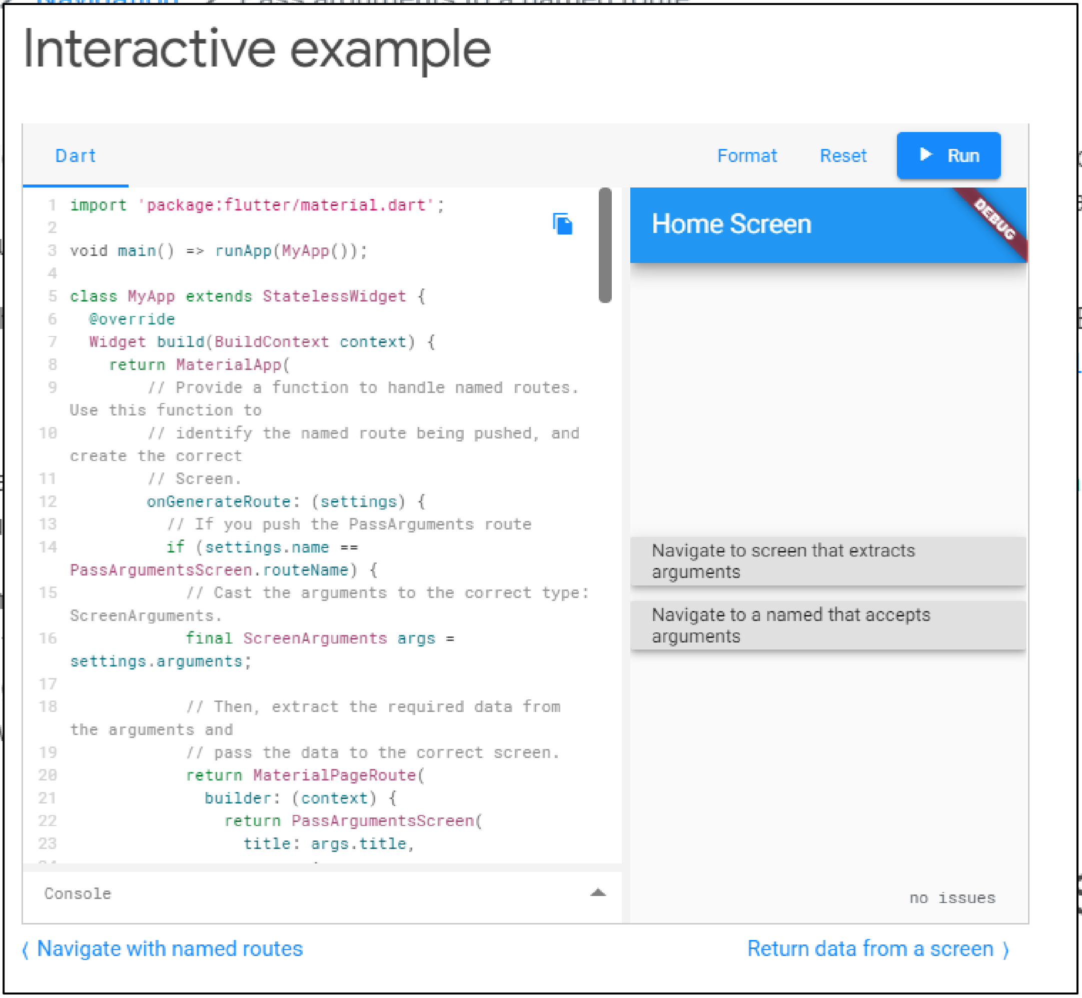
This recipe demonstra… …lRoute.of() and onGenerateRoute() u…

1. Define the argun…
2. Create a widget…
3. Register the wid…
4. Navigate to the…

1. Define… …s

First, define the arguments you need to pass to the new route. In this example, pass two pieces of data: The title of the screen

To pass both pieces of data, create a class that stores this information.



Interactive example

Dart   Format   Reset   ▶ Run

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

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      // Provide a function to handle named routes. Use this function to
      // identify the named route being pushed, and create the correct
      // Screen.
      onGenerateRoute: (settings) {
        // If you push the PassArguments route
        if (settings.name == PassArgumentsScreen.routeName) {
          // Cast the arguments to the correct type: ScreenArguments.
          final ScreenArguments args = settings.arguments;

          // Then, extract the required data from the arguments and
          // pass the data to the correct screen.
          return MaterialPageRoute(
            builder: (context) {
              return PassArgumentsScreen(
                title: args.title,
```

Console

Home Screen
DEBUG
Navigate to screen that extracts arguments
Navigate to a named that accepts arguments
no issues

⟨ Navigate with named routes   Return data from a screen ⟩

https://flutter.dev/docs/cookbook/navigation/navigate-with-arguments

Route functions

Abstract routes as resources within application

Subscribe on Youtube
Flutter Navigation Complete Named Routing Setup and Guide
Watch later
Share
TUTORIAL
NAVIGATE IN FLUTTER USING NAMED ROUTING
LIKE A PRO
Join me on Slack
View Code

Flutter Navigation Cheatsheet - A Guide to Named Routing

A simple guide that covers the setup and all navigation scenarios using named routing.

Written by **Dane Mackier**
Lead Developer and Software Architect

Flutter Navigation Cheatsheet - A Guide to Named Routing
https://www.filledstacks.com/post/flutter-navigation-cheatsheet-a-guide-to-named-routing/

Can use route functions ...

Clean Navigation in Flutter Using Generated Routes

Dane Mackier Follow
May 23, 2019 · 3 min read

```
static Route<dynamic> generateRoute(RouteSettings settings) {
    switch (settings.name) {
      case '/':
        return MaterialPageRoute(builder: (_) => Home());
      case '/feed':
        return MaterialPageRoute(builder: (_) => Feed());
      default:
        return MaterialPageRoute(
            builder: (_) => Scaffold(
                  body: Center(
                      child: Text('No route defined for
${settings.name}')),
                ));
    }
  }
```

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      onGenerateRoute: Router.generateRoute,
      initialRoute: homeRoute,
    );
  }
}
```

https://medium.com/flutter-community/clean-navigation-in-flutter-using-generated-routes-891bd6e000df

To associate names to views

Clean Navigation in Flutter Using Generated Routes

Dane Mackier Follow
May 23, 2019 · 3 min read

Can use a function to route according to context and settings (usually name)

```
static Route<dynamic> generateRoute(RouteSettings settings) {
    switch (settings.name) {
      case '/':
        return MaterialPageRoute(builder: (_) => Home());
      case '/feed':
        return MaterialPageRoute(builder: (_) => Feed());
      default:
        return MaterialPageRoute(
            builder: (_) => Scaffold(
                  body: Center(
                      child: Text('No route defined for
${settings.name}')),
                ));
    }
  }
```

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      onGenerateRoute: Router.generateRoute,
      initialRoute: homeRoute,
    );
  }
}
```

This function is given to MaterialApp as “onGenerateRoute” function

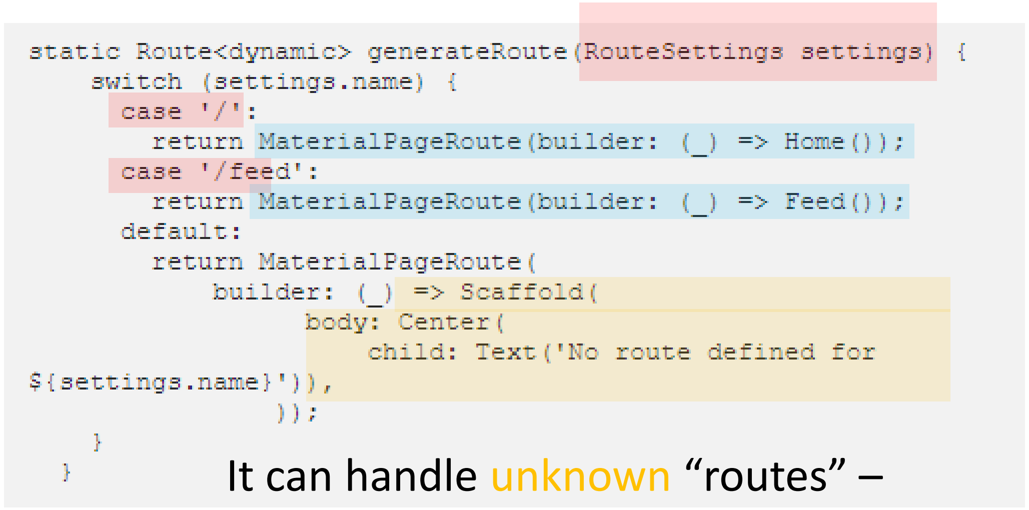
https://medium.com/flutter-community/clean-navigation-in-flutter-using-generated-routes-891bd6e000df

can use functions that solves the names

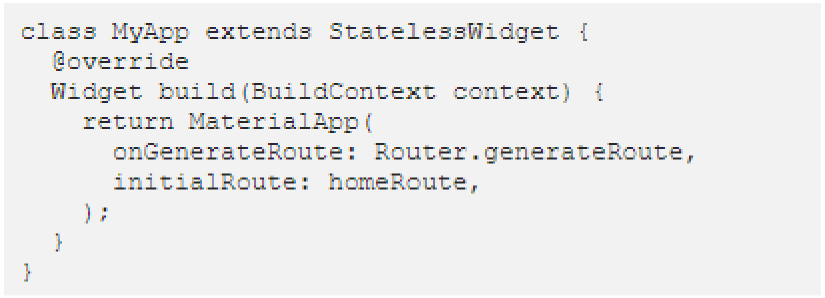
Clean Navigation in Flutter Using Generated Routes

Dane Mackier Follow
May 23, 2019 · 3 min read

It selects the route according to context and settings (usually name)



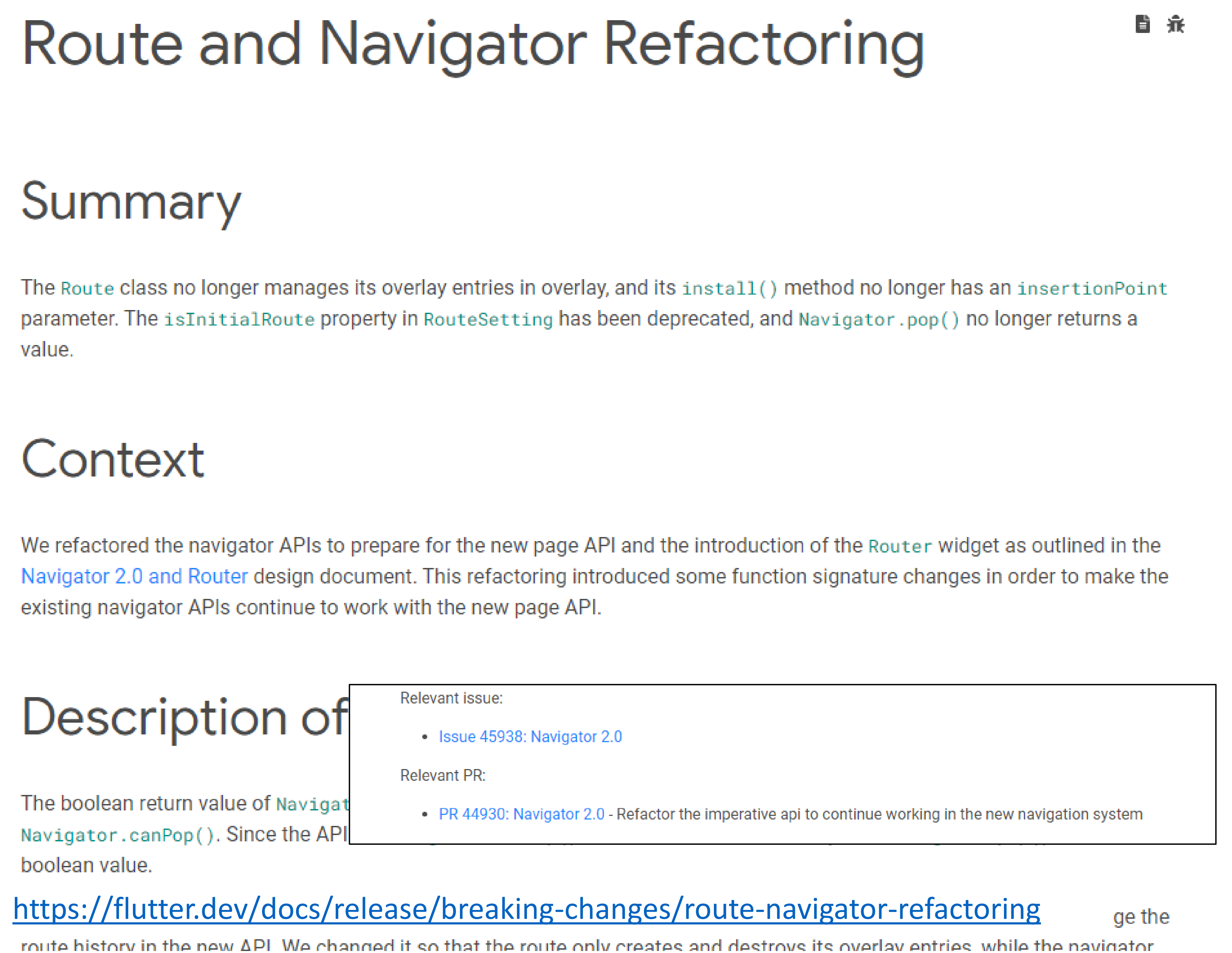
```
static Route<dynamic> generateRoute(RouteSettings settings) {
    switch (settings.name) {
      case '/':
        return MaterialPageRoute(builder: (_) => Home());
      case '/feed':
        return MaterialPageRoute(builder: (_) => Feed());
      default:
        return MaterialPageRoute(
            builder: (_) => Scaffold(
                  body: Center(
                      child: Text('No route defined for
${settings.name}')),
                ));
    }
  }
```



```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      onGenerateRoute: Router.generateRoute,
      initialRoute: homeRoute,
    );
  }
}
```

It can handle unknown "routes" – generating **error handling "widgets"**

https://medium.com/flutter-community/clean-navigation-in-flutter-using-generated-routes-891bd6e000df



Route and Navigator Refactoring

Summary

The `Route` class no longer manages its overlay entries in overlay, and its `install()` method no longer has an `insertionPoint` parameter. The `isInitialRoute` property in `RouteSetting` has been deprecated, and `Navigator.pop()` no longer returns a value.

Context

We refactored the navigator APIs to prepare for the new page API and the introduction of the `Router` widget as outlined in the Navigator 2.0 and Router design document. This refactoring introduced some function signature changes in order to make the existing navigator APIs continue to work with the new page API.

Description of

Relevant issue:

- Issue 45938: Navigator 2.0

Relevant PR:

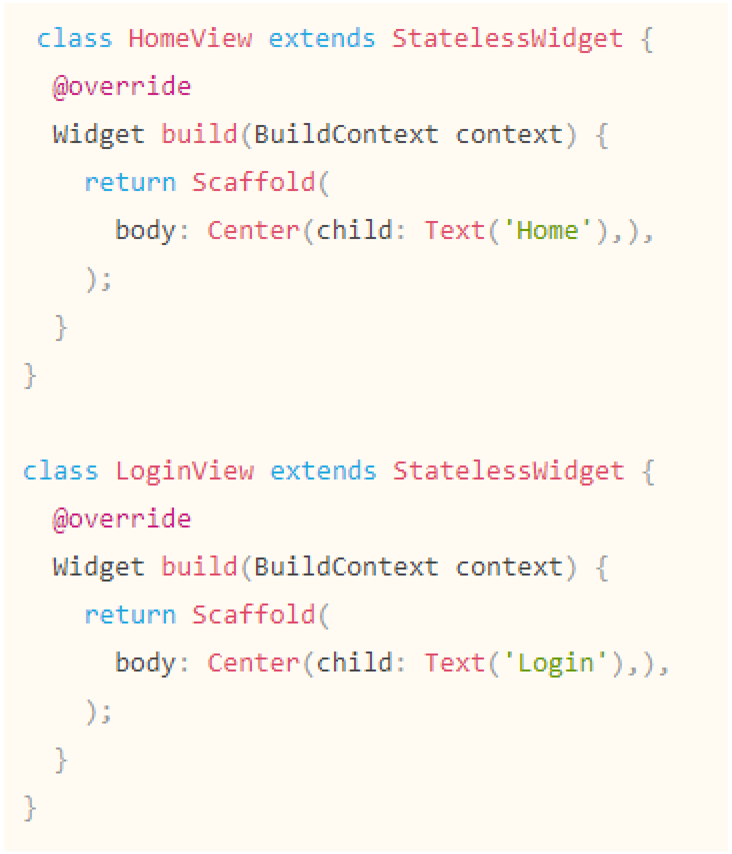
- PR 44930: Navigator 2.0 - Refactor the imperative api to continue working in the new navigation system

The boolean return value of `Navigat`
`Navigator.canPop()`. Since the API
boolean value.

https://flutter.dev/docs/release/breaking-changes/route-navigator-refactoring

ge the
route history in the new API. We changed it so that the route only creates and destroys its overlay entries, while the navigator

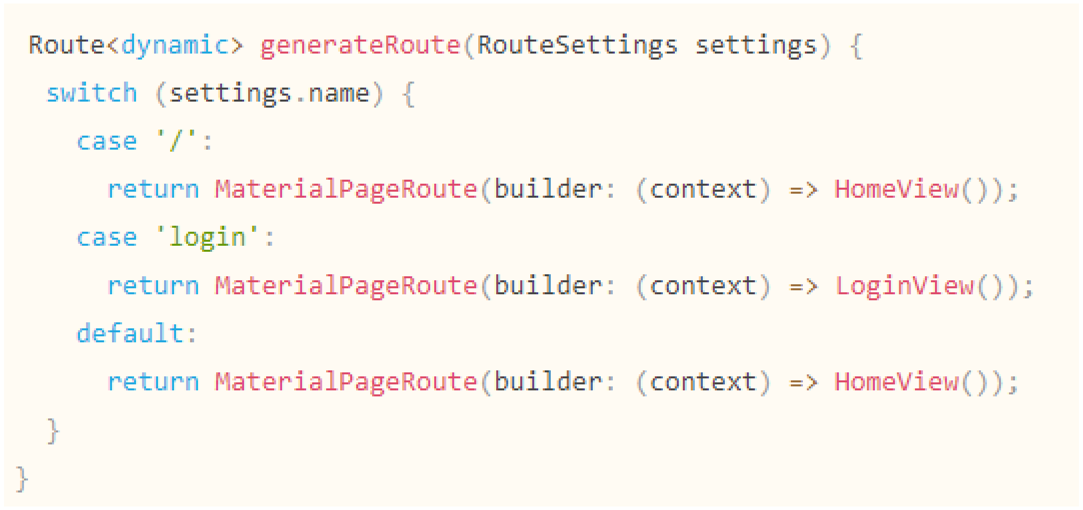
Two views



```
class HomeView extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(child: Text('Home'),),
    );
  }
}

class LoginView extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(child: Text('Login'),),
    );
  }
}
```

Flutter Navigation Cheatsheet - A Guide to Named Routing
https://www.filledstacks.com/post/flutter-navigation-cheatsheet-a-guide-to-named-routing/

A routing function



```
Route<dynamic> generateRoute(RouteSettings settings) {
  switch (settings.name) {
    case '/':
      return MaterialPageRoute(builder: (context) => HomeView());
    case 'login':
      return MaterialPageRoute(builder: (context) => LoginView());
    default:
      return MaterialPageRoute(builder: (context) => HomeView());
  }
}
```

Suggestion: use “global” string name

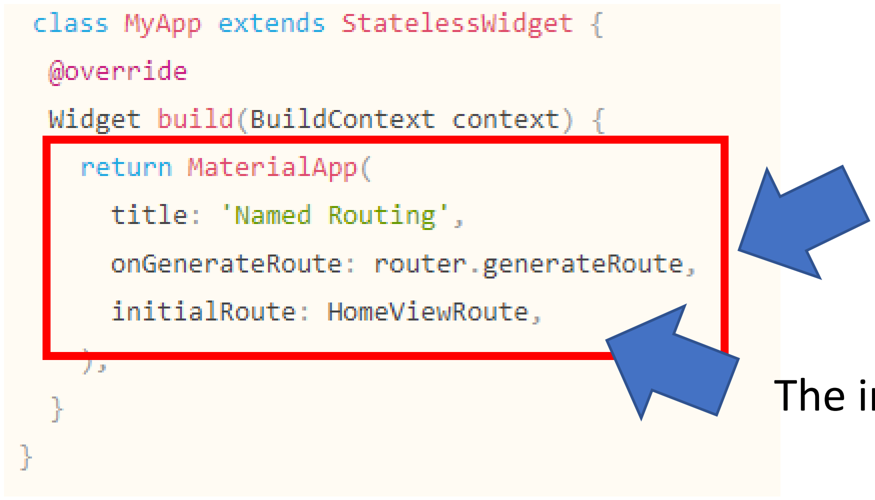
```
Route<dynamic> generateRoute(RouteSettings settings) {
 switch (settings.name) {
   case '/':
     return MaterialPageRoute(builder: (context) => HomeView());
   case 'login':
     return MaterialPageRoute(builder: (context) => LoginView());
   default:
     return MaterialPageRoute(builder: (context) => HomeView());
 }
}
```

```
   const String HomeViewRoute = '/';
 const String LoginViewRoute = 'login';
```

```
 switch (settings.name) {
  case HomeViewRoute:
     ...
  case LoginViewRoute:
     ...
}
```

https://www.filledstacks.com/post/flutter-navigation-cheatsheet-a-guide-to-named-routing/
https://www.youtube.com/watch?v=YXDFlpdpp3g&list=PLdTodMosi-BwEwlzjN6EyS1vwGXFo-UlK&index=5

Couple application and routing function



```
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Named Routing',
     onGenerateRoute: router.generateRoute,
     initialRoute: HomeViewRoute,
   );
 }
}
```

The function

The initial "screen"

Handling unknow routes: you decide…

```
class UndefinedView extends StatelessWidget {
 final String name;
 const UndefinedView({Key key, this.name}) : super(key: key);

 @override
 Widget build(BuildContext context) {
   return Scaffold(
     body: Center(
       child: Text('Route for $name is not defined'),
     ),
   );
 }
}
```

Define a "undefined view" with a message

Option 2

```
return MaterialApp(
 title: 'Named Routing',
 onGenerateRoute: router.generateRoute,
 onUnknownRoute: (settings) => MaterialPageRoute(
     builder: (context) => UndefinedView(
           name: settings.name,
         )),
 initialRoute: HomeViewRoute,
);
```

On the application

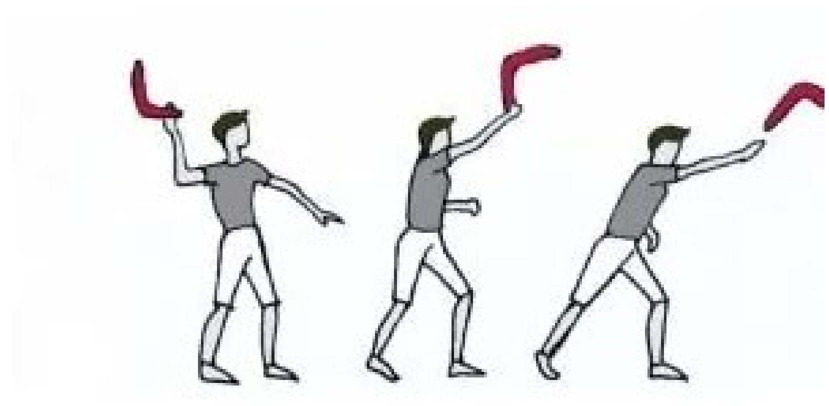
Option 1

```
switch (settings.name) {
  ...
  default:
    return MaterialPageRoute(builder: (context) => UndefinedView(name: settings.name,));
}
```

On the route function

Send and Receive Data across Screens

(some examples)

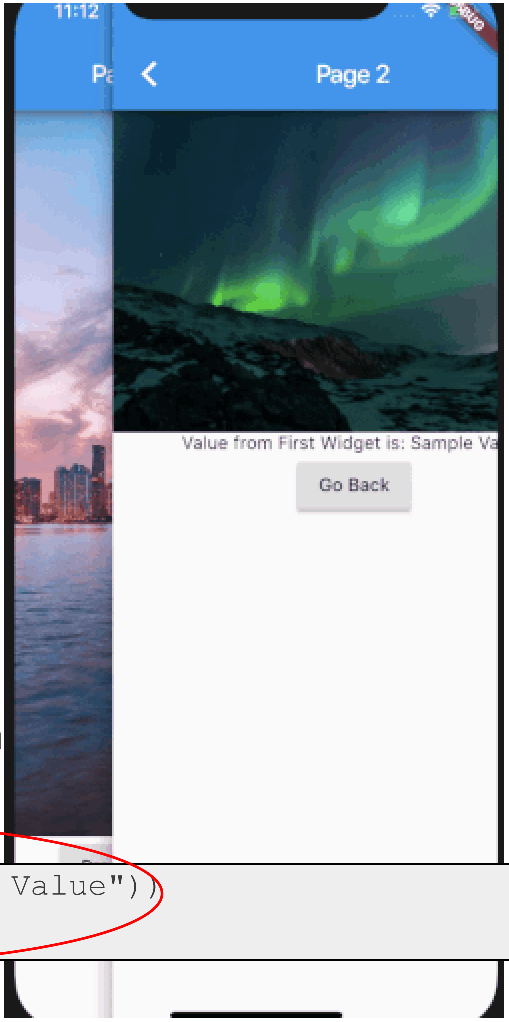


Pass data from window to other

- Pass data via the next widget via constructor
- Receive data from “popping” widget i.e. return

Pass data via the next widget via constructor

```
class FirstWidget extends StatelessWidget
{
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return Scaffold(
      appBar: AppBar(title: Text("Page 1"),),
      body: Column(children: <Widget>[
        Image.network("https://images.unsplash.com/photo-1535498730771-e73
        RaisedButton(child: Text("Press Here"),
        onPressed: () {
          Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => SecondWidget(value:"Sa
          );
        },
        )



  }


}
```



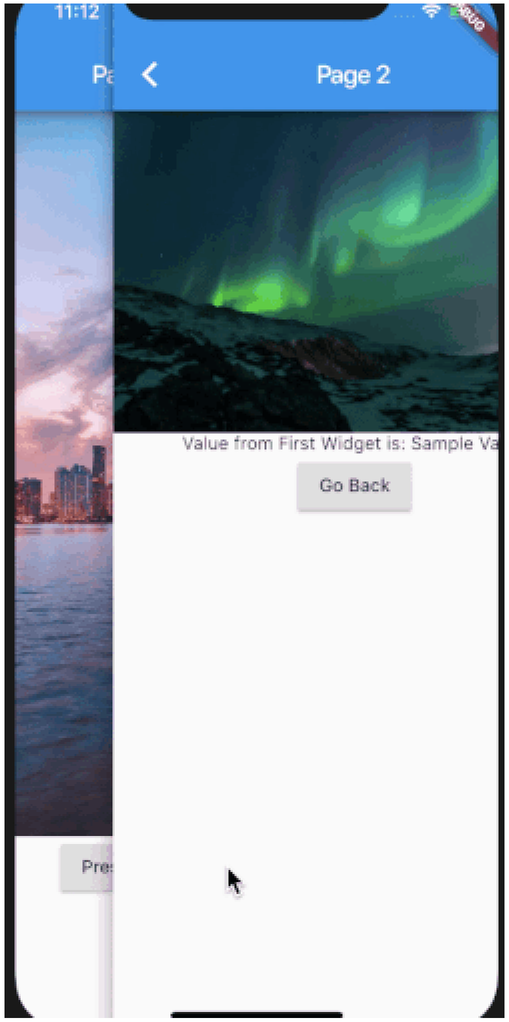
Passing data

```
MaterialPageRoute(builder: (context) => SecondWidget(value:"Sample Value"))
```

Pass data via the next widget via constructor

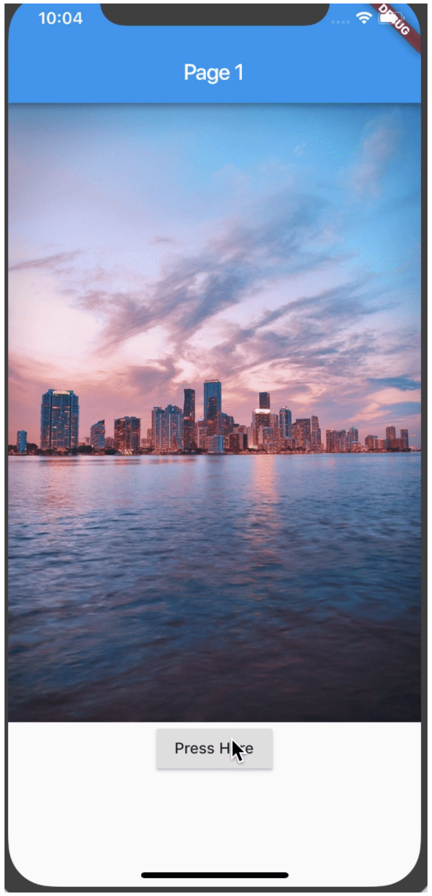
```
class SecondWidget extends StatelessWidget
{
  String value;
  SecondWidget({Key key, @required this.value}):super(key:key);
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return Scaffold(
      appBar: AppBar(title: Text("Page 2"),),
      body: Column(children: <Widget>[
        Image.network("https://www.w3schools.com/w3css/img_lights.jpg"),
        Text("Value from First Widget is: "+this.value),
        RaisedButton(child: Text("Go Back"),
            onPressed: () {
              Navigator.pop(context);
            }),
      ],)
    );
  }
}
```

Passing data with constructor



Receive data from “popping” widget

```
class FirstWidget extends StatelessWidget
{ String result;
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return Scaffold(
      appBar: AppBar(title: Text("Page 1"),),
      body: Builder(builder: (BuildContext context){

        return Column(children: <Widget>[
          Image.network("https://images.unsplash.com/photo-1535498730771-
          RaisedButton(child: Text("Press Here"),
              onPressed: (){
                displayValue(context);
              }
          ),
        ],);
      })

    );

  }
```



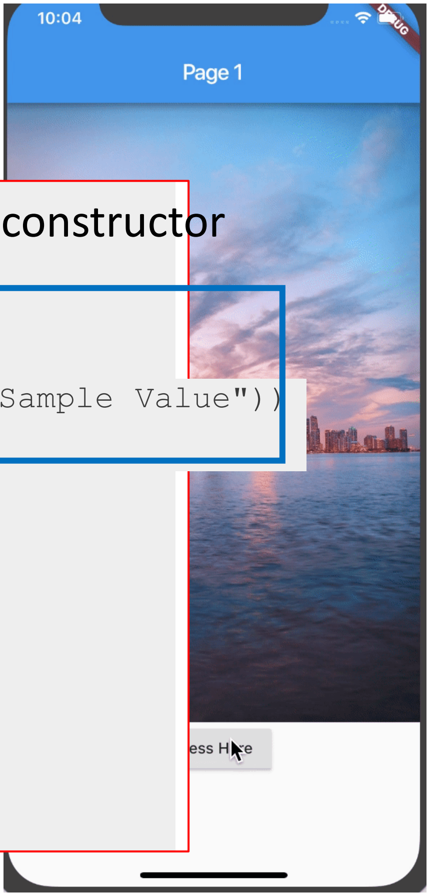
Receive data from “popping” widget i.e. return

```
class FirstWidget extends StatelessWidget
{ String result;
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
```

```
displayValue(BuildContext context) async {
   this.result = await Navigator.push(
      context,
     MaterialPageRoute(builder: (context) => SecondWidget(value:"Sample Value"))
  );
   Scaffold.of(context)
     ..removeCurrentSnackBar()
     ..showSnackBar(SnackBar(content: Text("$result")));
}

}
```

Pass “value” using constructor

```
    );

  }
```

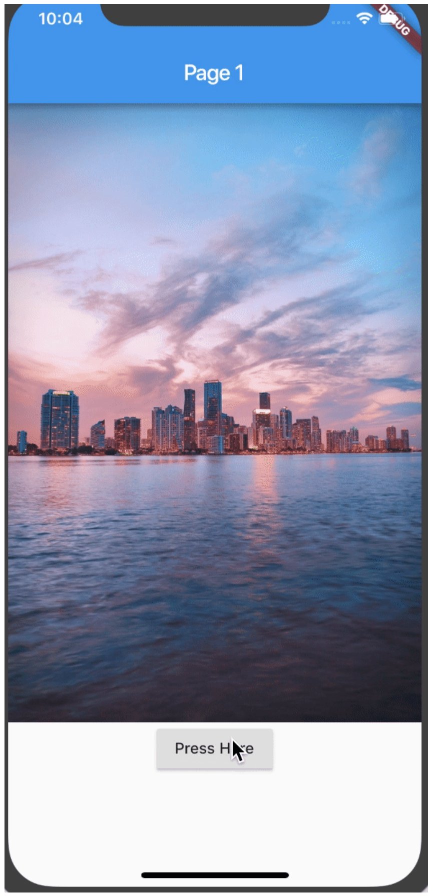


1

Receive data from “popping” widget

```
class SecondWidget extends StatelessWidget
{
  String value;
  SecondWidget({Key key, @required this.value}):super(key:key);
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return Scaffold(
      appBar: AppBar(title: Text("Page 2"),),
      body: Column(children: <Widget>[
        Image.network("https://www.w3schools.com/w3css/img_lights.jpg"),
        Text("Value from First Widget is: "+this.value),
        RaisedButton(child: Text("Go Back"),
            onPressed: () {
              Navigator.pop(context, "Sample Result Returned");
            }),
      ],)
    );
  }
```

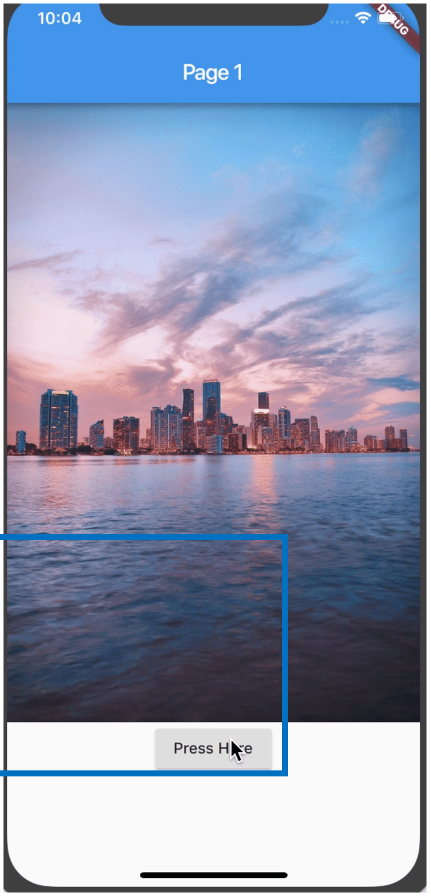
Receive using constructor



Receive data from “popping” widget

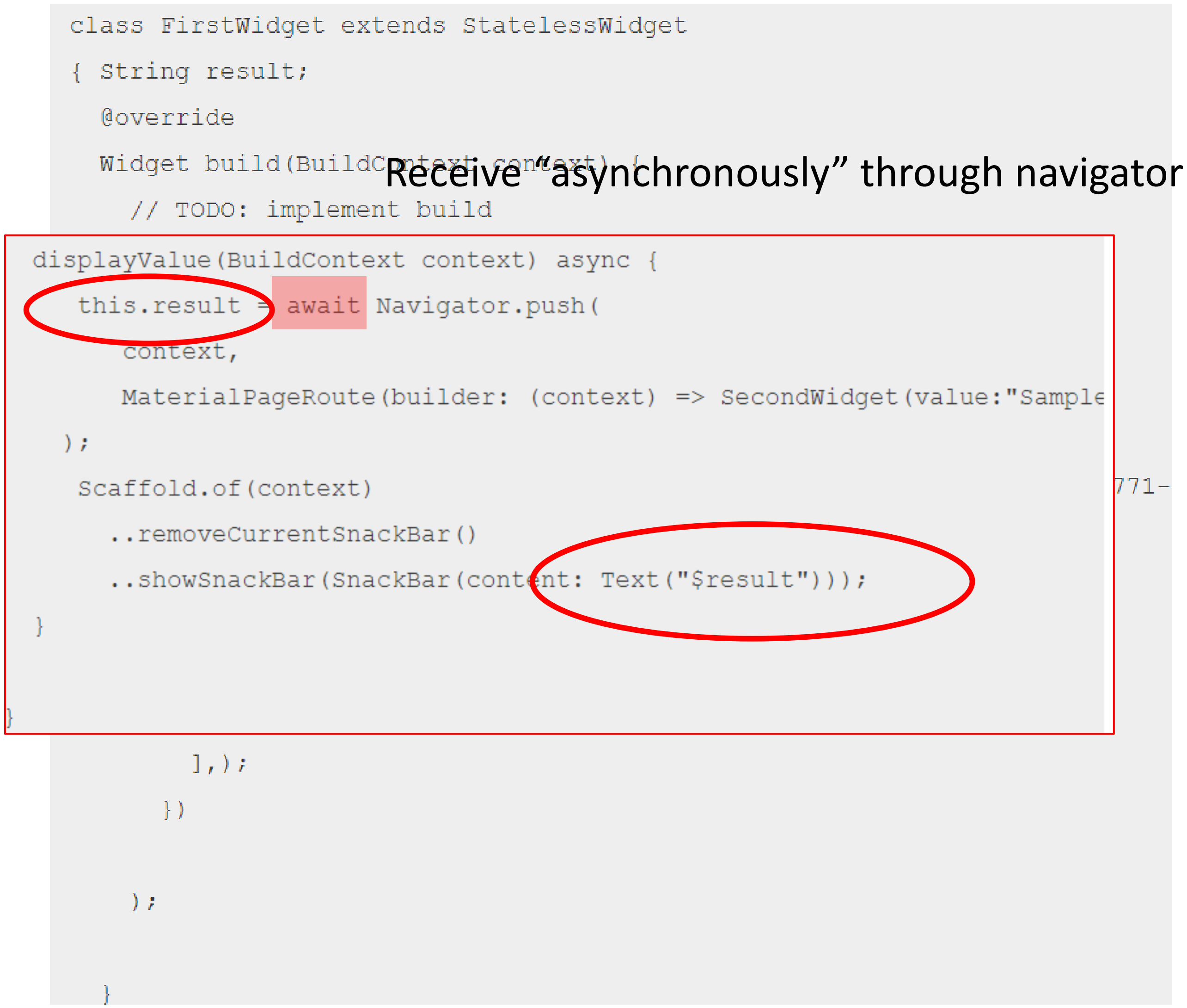
```
class SecondWidget extends StatelessWidget
{
  String value;
  SecondWidget({Key key, @required this.value}):super(key:key);
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
    return Scaffold(
      appBar: AppBar(title: Text("Page 2"),),
      body: Column(children: <Widget>[
        Image.network("https://www.w3schools.com/w3css/img_lights.jpg"),
        Text("Value from First Widget is: "+this.value),
        RaisedButton(child: Text("Go Back"),
            onPressed: () {
              Navigator.pop(context, "Sample Result Returned");
            }),
      ],)
    );
  }
```

Return through navigator

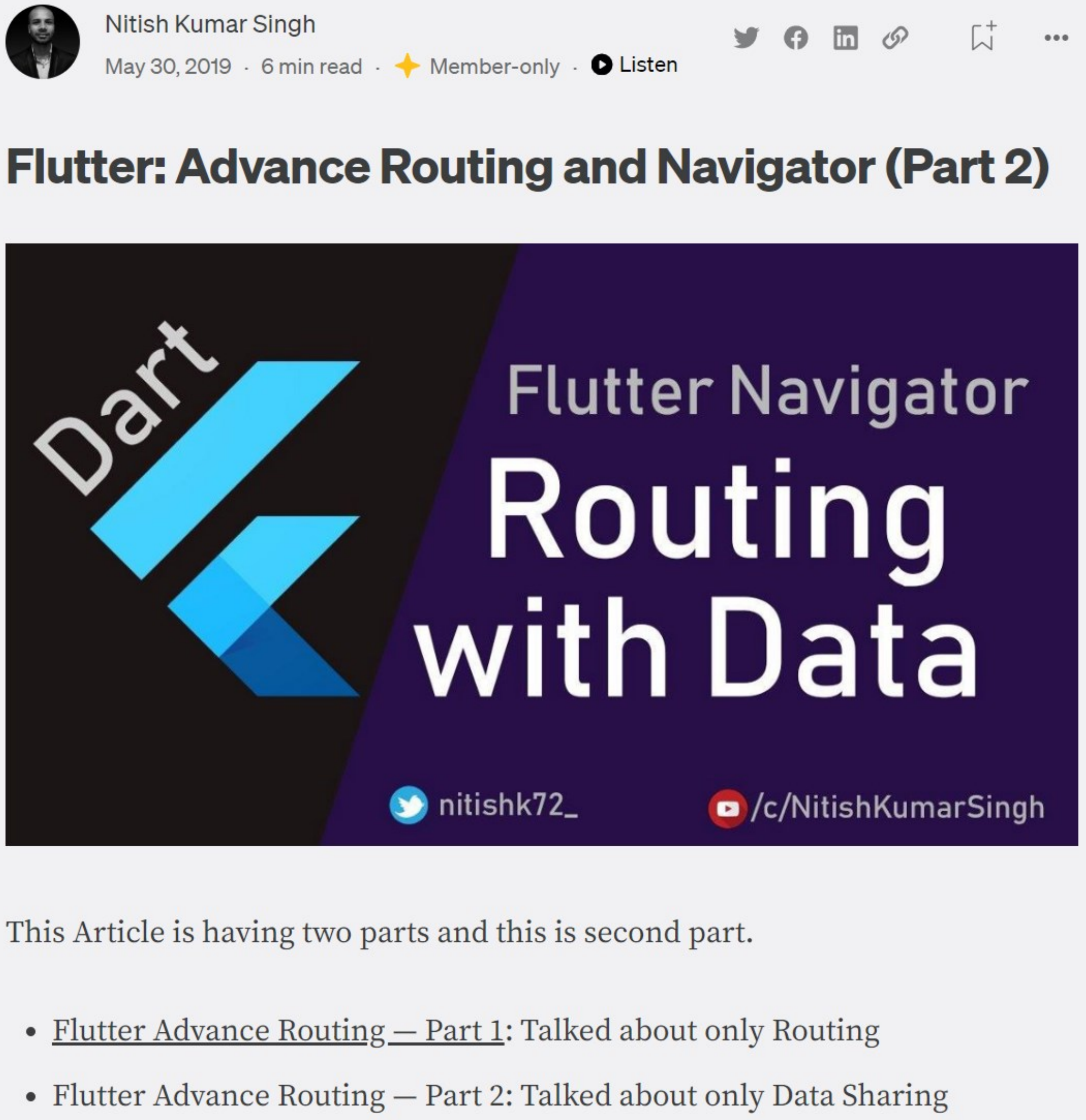


data from “popping” widget

Receive “asynchronously” through navigator



```
class FirstWidget extends StatelessWidget
{ String result;
  @override
  Widget build(BuildContext context) {
    // TODO: implement build
displayValue(BuildContext context) async {
   this.result = await Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => SecondWidget(value:"Sample
  );
   Scaffold.of(context)                                                  771-
     ..removeCurrentSnackBar()
     ..showSnackBar(SnackBar(content: Text("$result")));
 }

}
         ],);
       })

     );

   }
```



Nitish Kumar Singh

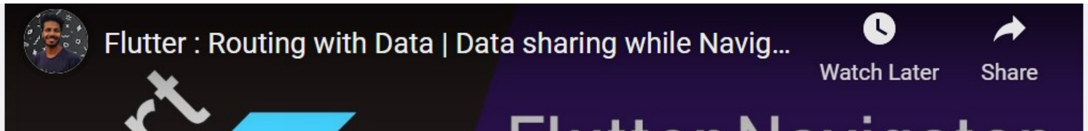
May 30, 2019 · 6 min read · Member-only · Listen

Flutter: Advance Routing and Navigator (Part 2)

This Article is having two parts and this is second part.

- Flutter Advance Routing — Part 1: Talked about only Routing
- Flutter Advance Routing — Part 2: Talked about only Data Sharing

https://nitishk72.medium.com/flutter-advance-routing-and-navigator-971c1e97d3d2



2

Passing objects: Simple Routing

```
class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Home'),
      ),
      body: Center(
        child: RaisedButton(
          onPressed: () {
            User user = new User(name: 'Nitish', age: 18);
            Route route =
              MaterialPageRoute(builder: (context) =>
SecondHome(user: user));
            Navigator.push(context, route);
          },
          child: Text('Second Home'),
        ),
      ),
    );
  }
}
```

```
class User {
  final String name;
  final int age;

  User({this.name, this.age});
}
```

https://blog.usejournal.com/flutter-advance-routing-and-navigator-971c1e97d3d2

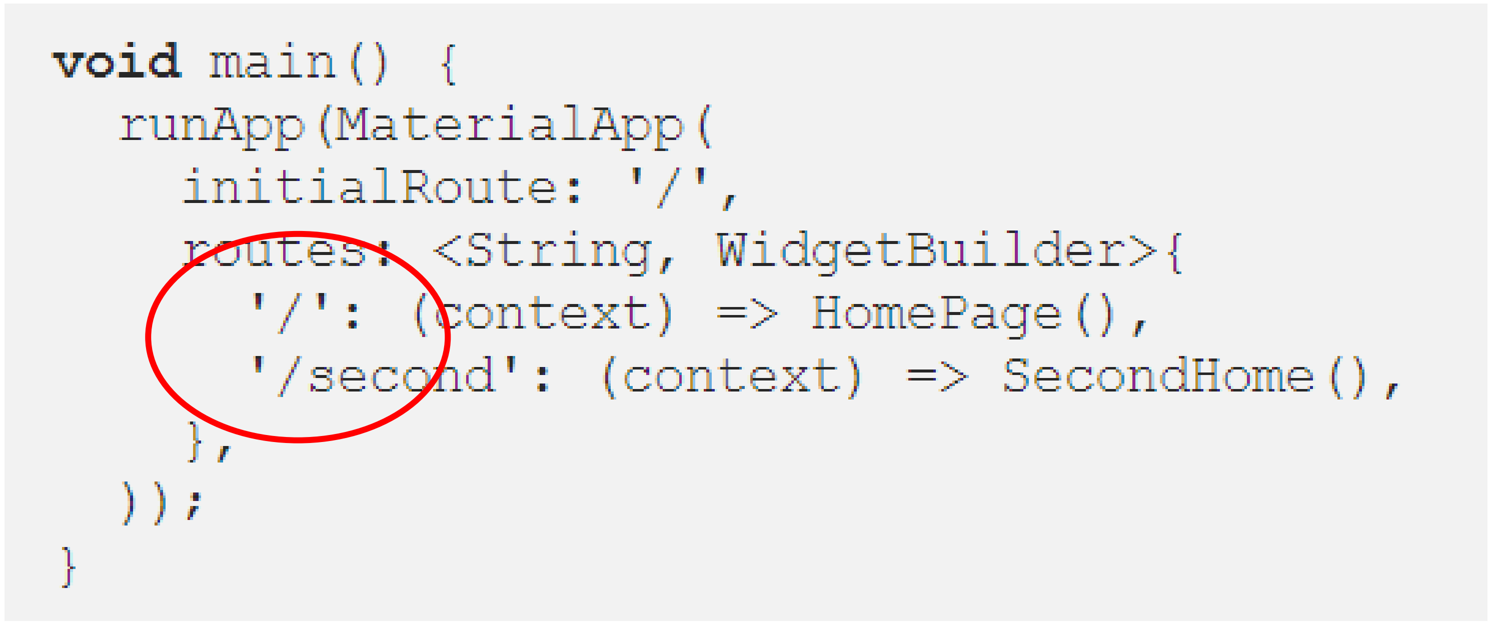
Passing objects: Simple Routing

```
class User {
  final String name;
  final int age;

  User({this.name, this.age});
}
```

```
class SecondHome extends StatelessWidget {
  final User user;

  SecondHome({this.user});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('${this.user.name} - ${this.user.age}'),
      ),
      body: Center(
        child: RaisedButton(
          onPressed: () {
            Navigator.pop(context);
          },
          child: Text('Go Back'),
        ),
      ),
    );
  }
}
```

https://blog.usejournal.com/flutter-advance-routing-and-navigator-971c1e97d3d2

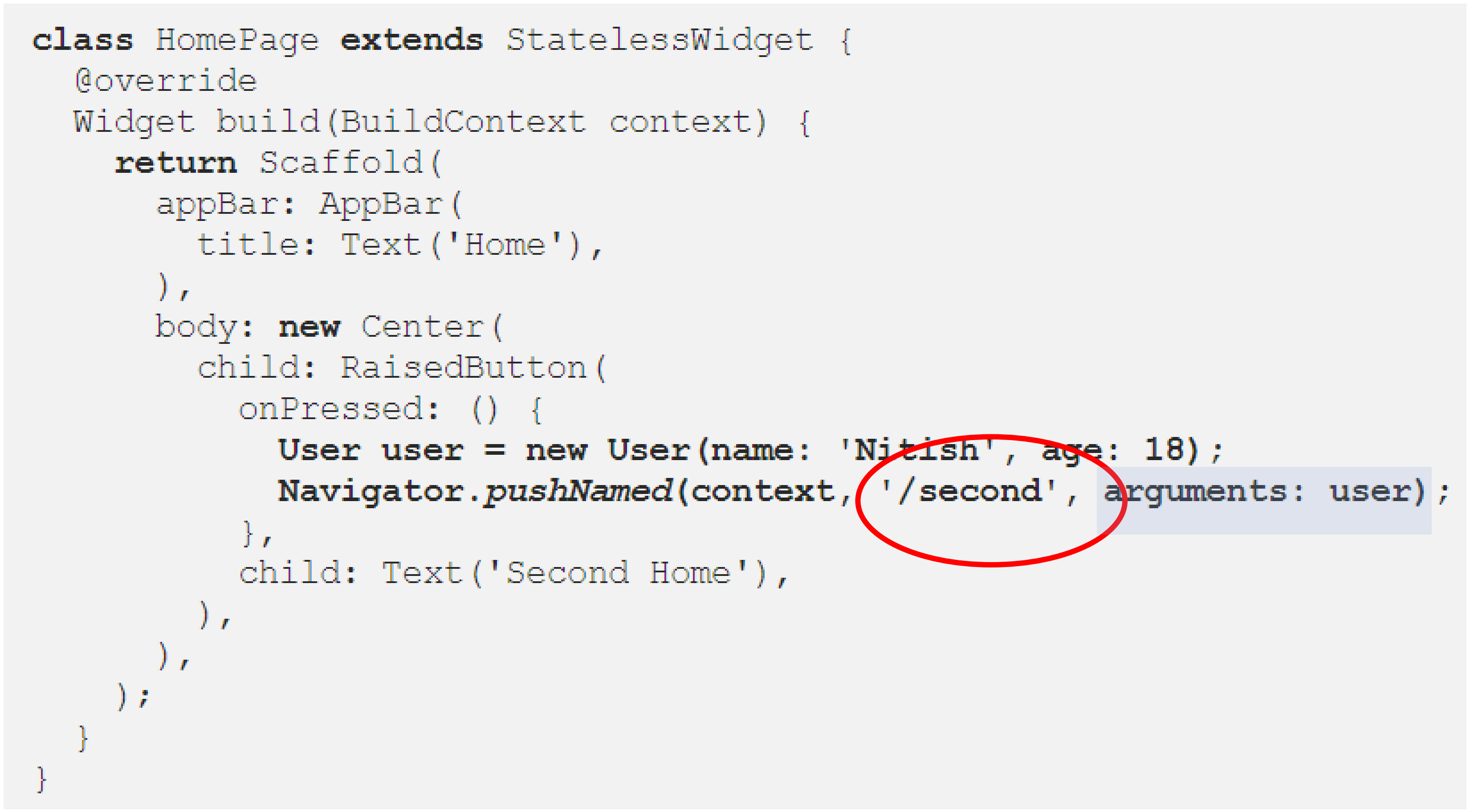
Passing objects: : Named Routing



```
void main() {
  runApp(MaterialApp(
    initialRoute: '/',
    routes: <String, WidgetBuilder>{
      '/': (context) => HomePage(),
      '/second': (context) => SecondHome(),
    },
  ));
}
```

2

Passing objects: Named Routing



```
class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Home'),
      ),
      body: new Center(
        child: RaisedButton(
          onPressed: () {
            User user = new User(name: 'Nitish', age: 18);
            Navigator.pushNamed(context, '/second', arguments: user);
          },
          child: Text('Second Home'),
        ),
      ),
    );
  }
}
```

Passing objects: Named Routing

```
class SecondHome extends StatelessWidget {
  User user;

  @override
  Widget build(BuildContext context) {
    RouteSettings settings = ModalRoute.of(context).settings;
    user = settings.arguments;
    return Scaffold(
      appBar: AppBar(
        title: Text('${this.user.name} - ${this.user.age}'),
      ),
      body: new Center(
        child: RaisedButton(
          onPressed: () {
            Navigator.pop(context);
          },
          child: Text('Go Back'),
        ),
      ),
    );
  }
}
```

Passing objects: onGenerateRoute

```
void main() {
  runApp(
    MaterialApp(
      home: HomePage(),
      onGenerateRoute: (RouteSettings settings) {
        switch (settings.name) {
          case '/':
            return MaterialPageRoute(builder: (context) =>
HomePage());
            break;
          case '/second':
            User user = settings.arguments;
            return MaterialPageRoute(
              builder: (context) => SecondHome(user: user),
            );
            break;
        }
      },
    ),
  );
}
```

2

Passing objects: onGenerateRoute

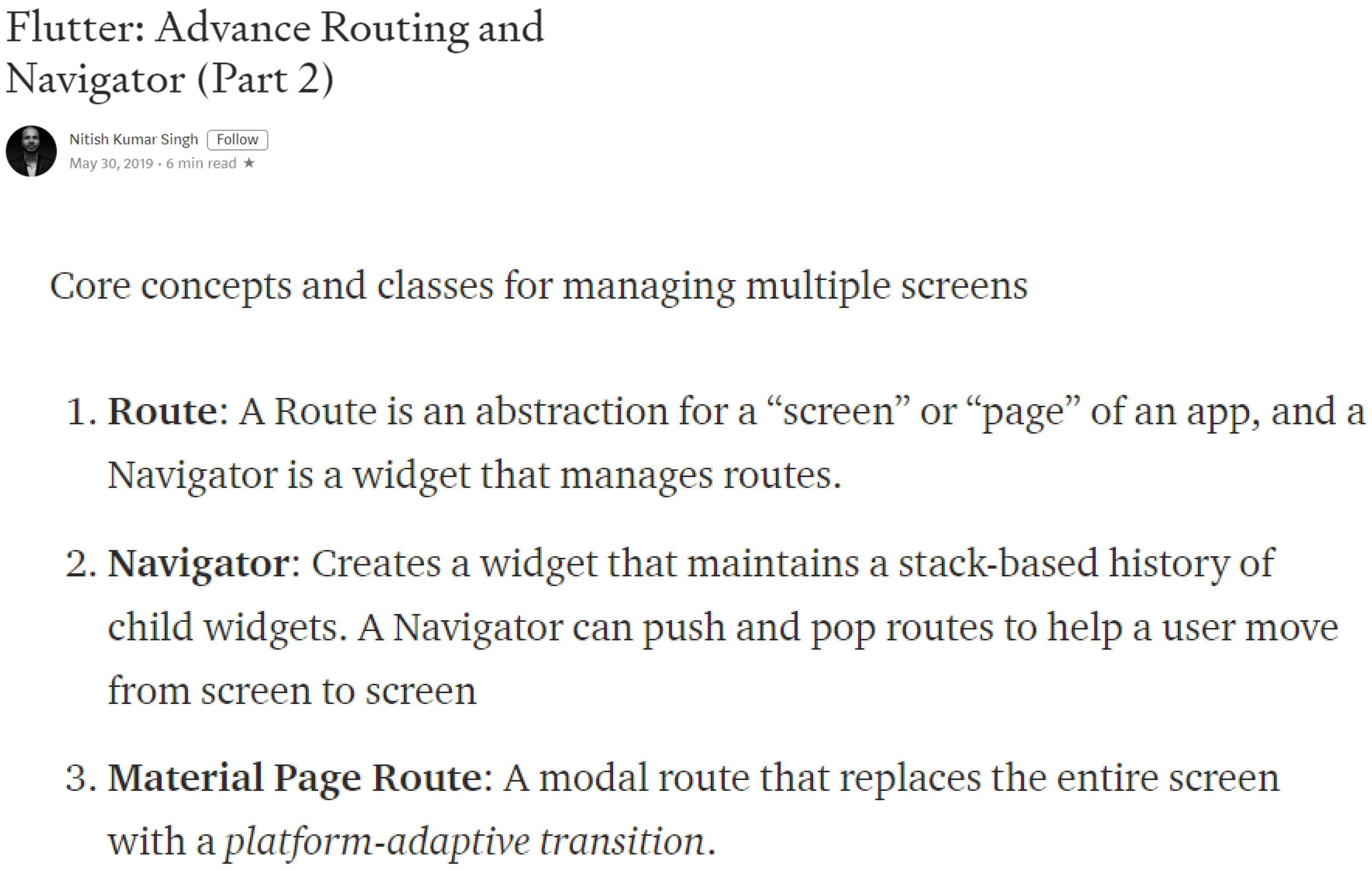
```
class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Home'),
      ),
      body: new Center(
        child: RaisedButton(
          onPressed: () {
            User user = new User(name: 'Nitish', age: 18);
            Navigator.pushNamed(context, '/second', arguments: user);
          },
          child: Text('Second Home'),
        ),
      ),
    );
  }
}
```

```
class SecondHome extends StatelessWidget {
  final User user;

  SecondHome({this.user});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('${this.user.name} - ${this.user.age}'),
      ),
      body: new Center(
        child: RaisedButton(
          onPressed: () {
            Navigator.pop(context);
          },
          child: Text('Go Back'),
        ),
      ),
    );
  }
}
```

Advanced routing

Several interesting examples

- Flutter: Advance Routing and Navigator (Part 1 & 2) - may 2019
  - Flutter Advance Routing — Part 1: Talked about only Routing
  - Flutter Advance Routing — Part 2: Talked about only Data Sharing
- Flutter navigation — routing made easy - jan2020
  - https://itnext.io/flutter-navigation-routing-made-easy-816ddf9e2857



Flutter: Advance Routing and Navigator (Part 2)

Nitish Kumar Singh Follow
May 30, 2019 · 6 min read ★

Core concepts and classes for managing multiple screens

1. **Route**: A Route is an abstraction for a “screen” or “page” of an app, and a Navigator is a widget that manages routes.
2. **Navigator**: Creates a widget that maintains a stack-based history of child widgets. A Navigator can push and pop routes to help a user move from screen to screen
3. **Material Page Route**: A modal route that replaces the entire screen with a *platform-adaptive transition*.

https://blog.usejournal.com/flutter-advance-routing-and-navigator-971c1e97d3d2

The END