UNIVERSITY OF PADOVA
DEPARTMENT OF INFORMATION ENGINEERING

Biomedical Wearable Technologies
for Healthcare and Wellbeing

Navigation

A.Y. 2022-2023

Giacomo Cappon

DEPARTMENT OF
INFORMATION
ENGINEERING
UNIVERSITY OF PADOVA

Outline

- **Asynchrony**
- Navigator
- Navigate to a new screen and back
- Named routes
- Passing argument to a named routes
- Returning an argument from a named route

- Exercise
- Homework
- Resources

New concept: asynchrony

- Let's learn something (I believe) new.
- Dart (and Flutter) is full of asynchronous functions: they return after doing **something** possibly time consuming without waiting for that **something** to complete
- Common asynchronous operations:
  - Fetching data over the net
  - Writing/Reading data from a database
  - Load and show an image stored within the phone
- This is a problem because this
  - ```
    fetchDataFromFacebook(); // <-- asynchronous stuff
    print('Done');
    ```

  Can possibly print 'Done' before actually finishing fetching data!
- We need to learn how to manage asynchronous code in a synchronized fashion!
    ```

Key terms

- **synchronous operation**: A synchronous operation blocks other operations from executing until it completes.
- **synchronous function**: A synchronous function only performs synchronous operations.
- **asynchronous operation**: Once initiated, an asynchronous operation allows other operations to execute before it completes.
- **asynchronous function**: An asynchronous function performs at least one asynchronous operation and can also perform synchronous operations.

Future

- Dart manages asynchrony using the **`Future`** class

- A future (lower case "f") is an instance of the `Future` class. A future represents the result of an asynchronous operation, and can have two states: uncompleted or completed.

  - **Uncompleted**: When you call an asynchronous function, it returns an uncompleted future. That future is waiting for the function's asynchronous operation to finish or to throw an error.

  - **Completed**:
    - **With a value**: A future of type `Future<T>` completes with a value of type `T`. For example, a future with type `Future<String>` produces a string value. If a future doesn't produce a usable value, then the future's type is `Future<void>`.

    - **With an error**: If the asynchronous operation performed by the function fails for any reason, the future completes with an error.

Future (wrong example)

```
void fetchUserOrder() {
  Future.delayed(const  Duration(seconds: 2),
() => print('Large Latte'));
}//fetchUserOrder

void main() {

  print('Fetching user order...');
  fetchUserOrder();
  print('Done');

}//main
```

The function is doing some asynchronous stuff.

Note: main is an asynchronous function now

Note: 'Done' will be print before 'Large latte'. How to fix this?

Full example in lab_06-navigation/01-introducing_futures.dart

Async and Await

- The **`async`** and **`await`** keywords provide a declarative way to define asynchronous functions and use their results. Remember these two basic guidelines when using `async` and `await`:

1. To define an asynchronous function, add **`async`** before the function body and wrap its return type in a `Future`.

2. The **`await`** keyword is used to wait for the result of an asynchronous function before going on and works only inside asynchronous functions.

Fixing the main function

- Let's then fix the main function:
  - First, add the `async` keyword before the function body

```
void main() async {}
```

  - Wrap the return type in a `Future`:

```
Future<void> main() async {}
```

- Now that you have a correctly defined async function, you can use the `await` keyword to wait for a future to complete:

```
await fetchUserOrder();
```

Fixing the fetchUserOrder function

- To fix the `fetchUserOrder` function we can proceed in a similar way
  - First, add the `async` keyword before the function body

```
void fetchUserOrder() async {}
```

  - Wrap the return type in a `Future:`

```
Future<void> fetchUserOrder() async {}
```

  - Then, `await` the end of the asynchronous operation:

```
await Future.delayed...
```

Future (correct example)

```
Future<void> fetchUserOrder() async {
  await Future.delayed(const
      Duration(seconds: 2), () =>
      print('Large Latte'));
}//fetchUserOrder

Future<void> main() async{

  print('Fetching user order...');
  await fetchUserOrder();
  print('Done');

}//main
```

Note: Now 'Done' will be print AFTER 'Large latte'.

Full example in lab_06-navigation/02-async_and_await.dart

Outline

- Asynchrony
- **Navigator**
- Navigate to a new screen and back
- Named routes
- Passing argument to a named routes
- Returning an argument from a named route

- Exercise
- Homework
- Resources

Navigator

- In general, apps are made of multiple screens (called **routes**)
- How to navigate through routes?
- How to pass things to routes and get values back from them?
- `Navigator` is a special class that allows to manage all of this

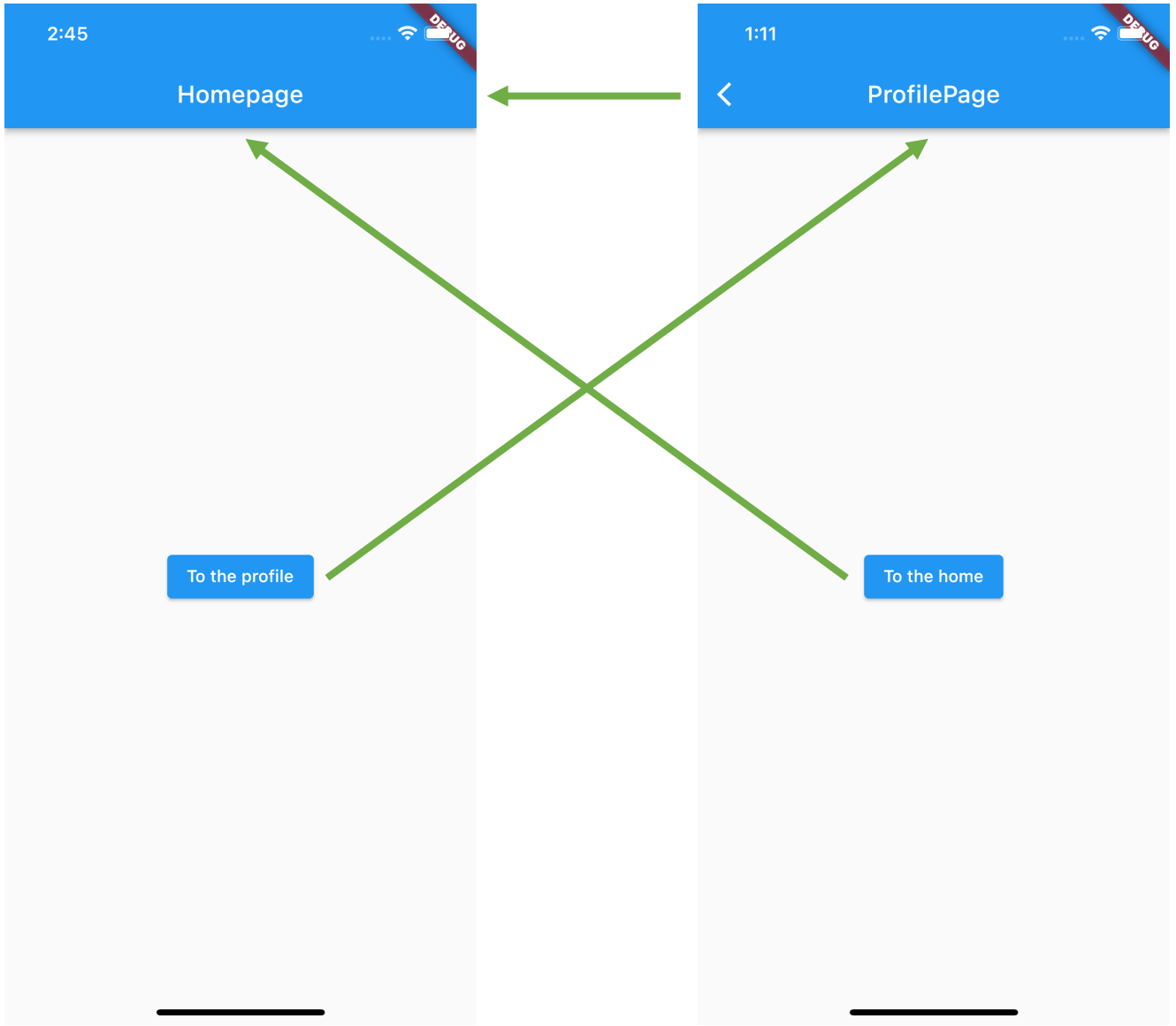
Outline

- Asynchrony
- Navigator
- **Navigate to a new screen and back**
- Named routes
- Passing argument to a named routes
- Returning an argument from a named route

- Exercise
- Homework
- Resources

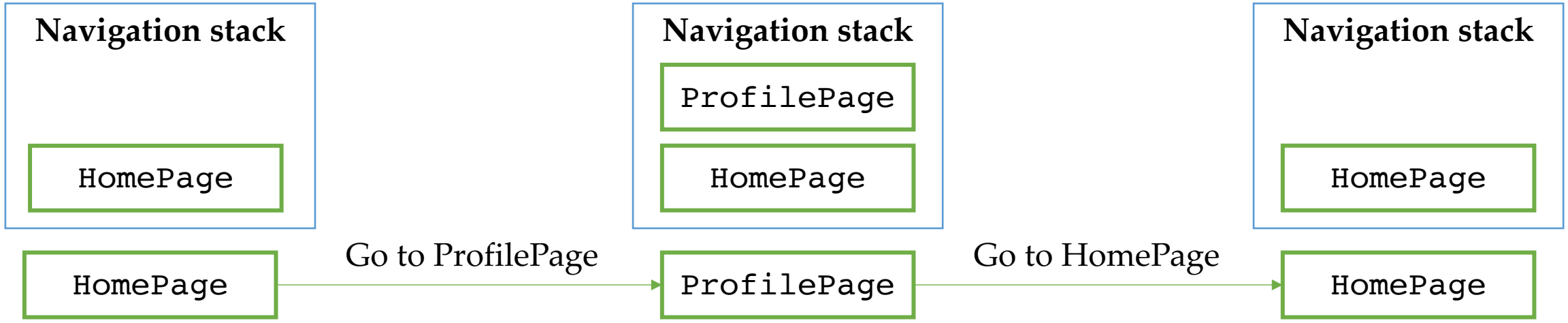
Navigator basics

- First let's see how to move between two routes
- We will start from creating a simple two-routes app where the first route will act as homepage and the second will represent the route that will ideally contain the info on the user profile.
- When the user taps the button on the homepage it will be directed to the profile page and viceversa



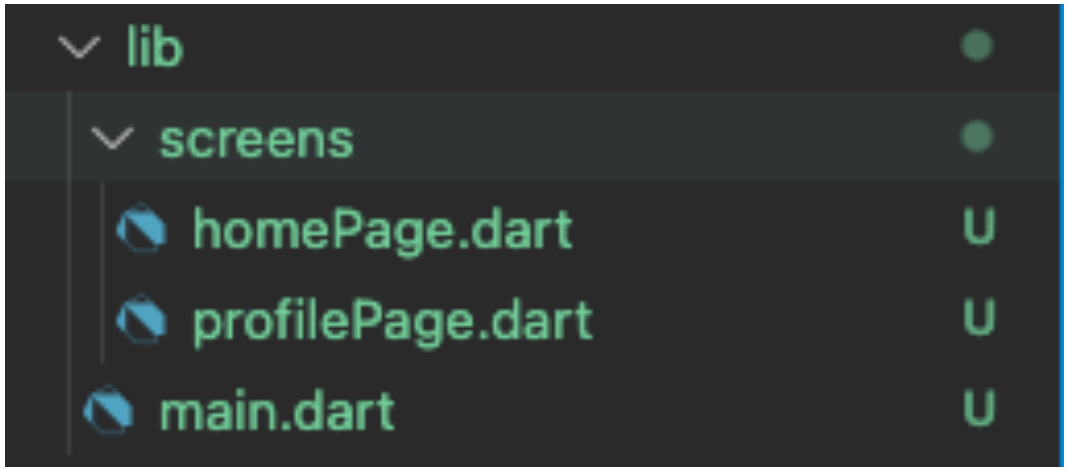
Navigator rationale

- `Navigator` is in charge of managing the navigation through the app
- To do so, `Navigator` uses a **stack-like structure**. The user sees the "top" of the stack
- When you go to a new route, you are "pushing" it into the stack
- When you go back, you are "popping" the route out of the `Navigator`



Navigator basics - Preparation

- Create a new project called ‘there_and_back_again’
- Create the `lib/screens/` folder
- Create two files in the `lib/screens/` folder just created and rename them as ’homePage.dart’ and ‘profilePage.dart’
- The project `lib` folder should look like this:



Navigator basics – homePage.dart boilerplate

```
import 'package:flutter/material.dart';
class HomePage extends StatelessWidget {
  const HomePage({Key? key}) :
super(key: key);
  static const routename = 'Homepage';
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(HomePage.routename),
      ),
      ...
```

```
...
body: Center(
        child: ElevatedButton(
          child: Text('To the profile'),
          onPressed: () {
            //TODO: implement the
navigation
          },
        ),
      ),
    );
  } //build
} //HomePage
```

Navigator basics – profilePage.dart boilerplate

```
import 'package:flutter/material.dart';
class ProfilePage extends StatelessWidget {
  const ProfilePage({Key? key}) :
super(key: key);
  static const routename = 'ProfilePage';
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(HomePage.routename),
      ),
      ...
```

```
...
body: Center(
        child: ElevatedButton(
          child: Text('To the home'),
          onPressed: () {
            //TODO: implement the
navigation
          },
        ),
      ),
    );
  } //build
} //ProfilePage
```

Navigator basics – main.dart boilerplate

```
import 'package:flutter/material.dart';
import
'package:there_and_back_again/screens/homepage.dart';


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


class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: HomePage(),
    );
  } //build
}//MyApp
```

Navigator basics – push and pop

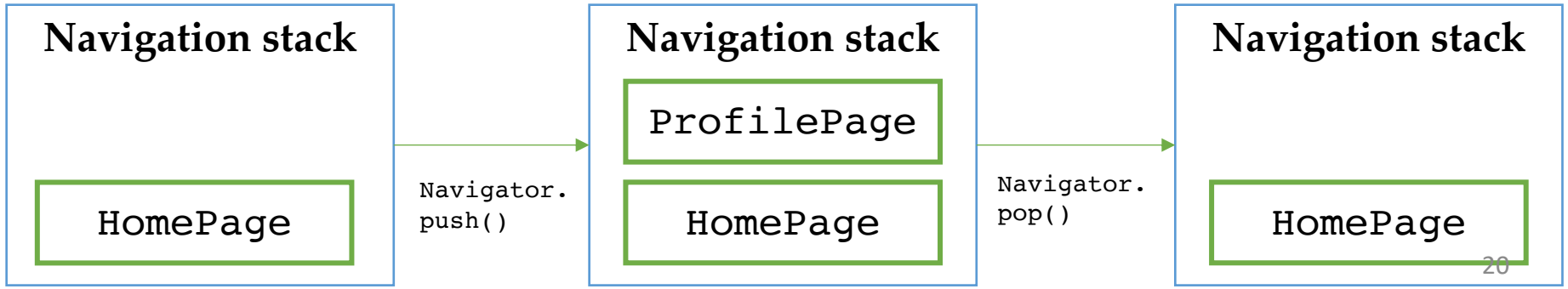
- To go to the ProfilePage route, simply invoke `Navigator.push()`:

```
...
onPressed: () {
  Navigator.push(context, MaterialPageRoute(builder: (context) => ProfilePage()));
},
...
```

Current `BuildContext` — The new `MaterialPageRoute` to be pushed into the stack

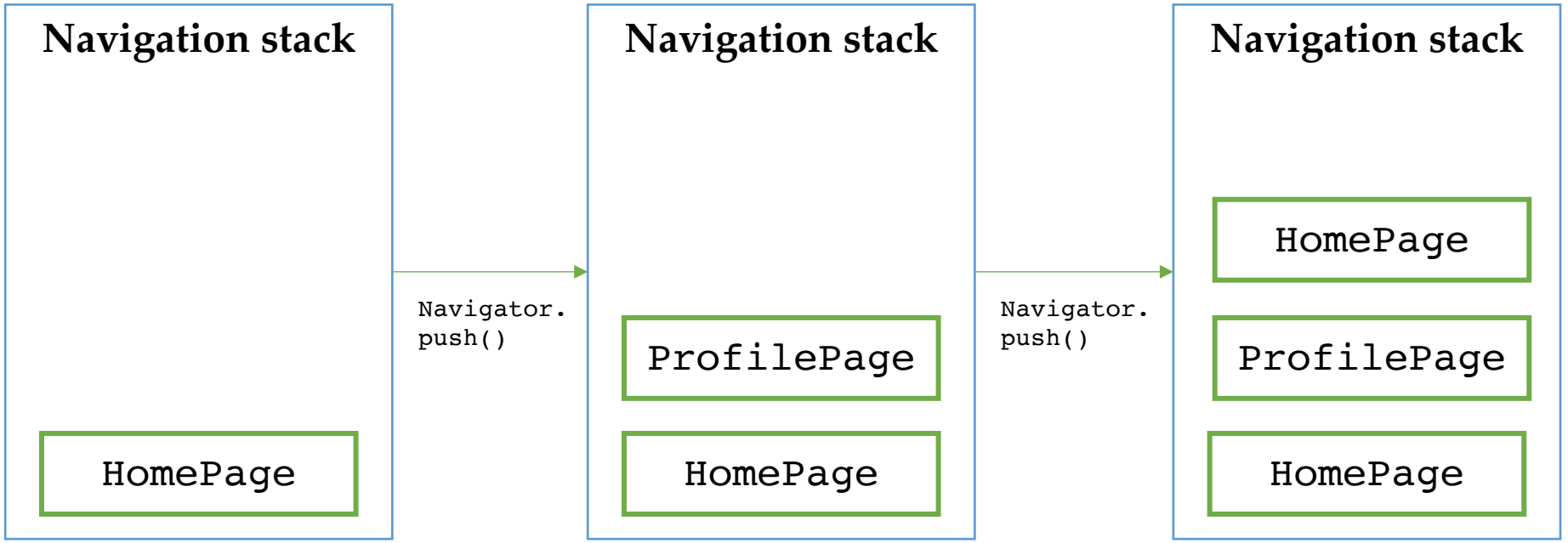
- To pop the ProfilePage route, simply invoke `Navigator.pop()`:

```
...
onPressed: () {
  Navigator.pop();
},
...
```



Navigator basics – push and pop

- Note that you could have used `Navigator.push()` to go back to the `HomePage` but this would have been result:



- Very messy situation. The stack will grow indefinetely

Outline

- Asynchrony
- Navigator
- Navigate to a new screen and back
- **Named routes**
- Passing argument to a named routes
- Returning an argument from a named route

- Exercise
- Homework
- Resources

Another approach: Named routes

- An alternative approach to `Navigator.push()` is `Navigator.pushNamed()`
- This solution consists of **associating names to each route** and use the names for navigation
- My personal opinion: this is a **cleaner approach** that leads to better, more readable code
- Let's see how to go for this approach

Named navigation – Preparation

- If you want to implement this approach, you need to specify, from the beginning, the name of each route.

- This is done via the `initialRoute` and `routes` named parameters of `MaterialApp`:

```
MaterialApp(
  initialRoute: '/',
  routes: {
    '/' : (context) => HomePage(),
    '/profile/': (context) => ProfilePage(),
  },
);
```

This specifies the app entry point

This maps names to the corresponding routes within the app

Named navigation – pushNamed

- To go to the `ProfilePage` route, now you can invoke `Navigator.pushNamed()`:

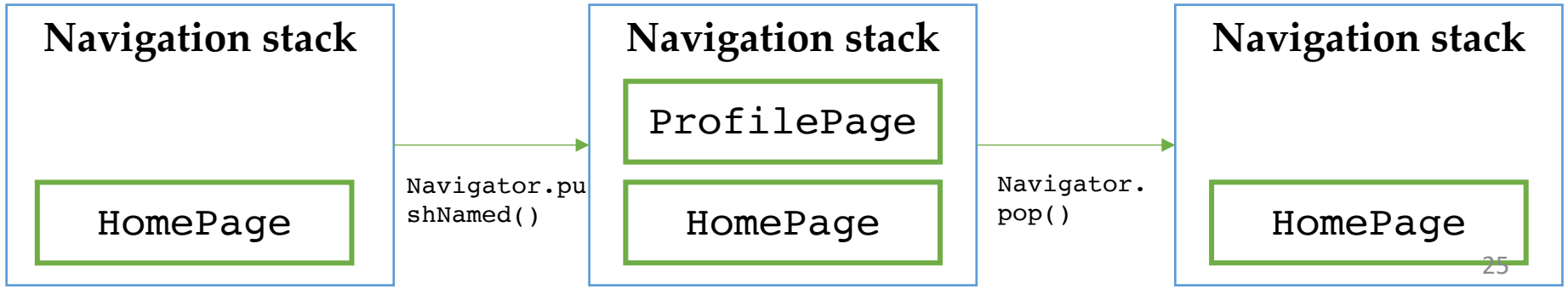
```
...
onPressed: () {
  Navigator.pushNamed(context,'/profile/');
},
...
```

Current `BuildContext`

The name of the route to be pushed into the stack

- To pop the `ProfilePage` route, you can still use `Navigator.pop()`:

```
...
onPressed: () {
  Navigator.pop();
},
...
```



Outline

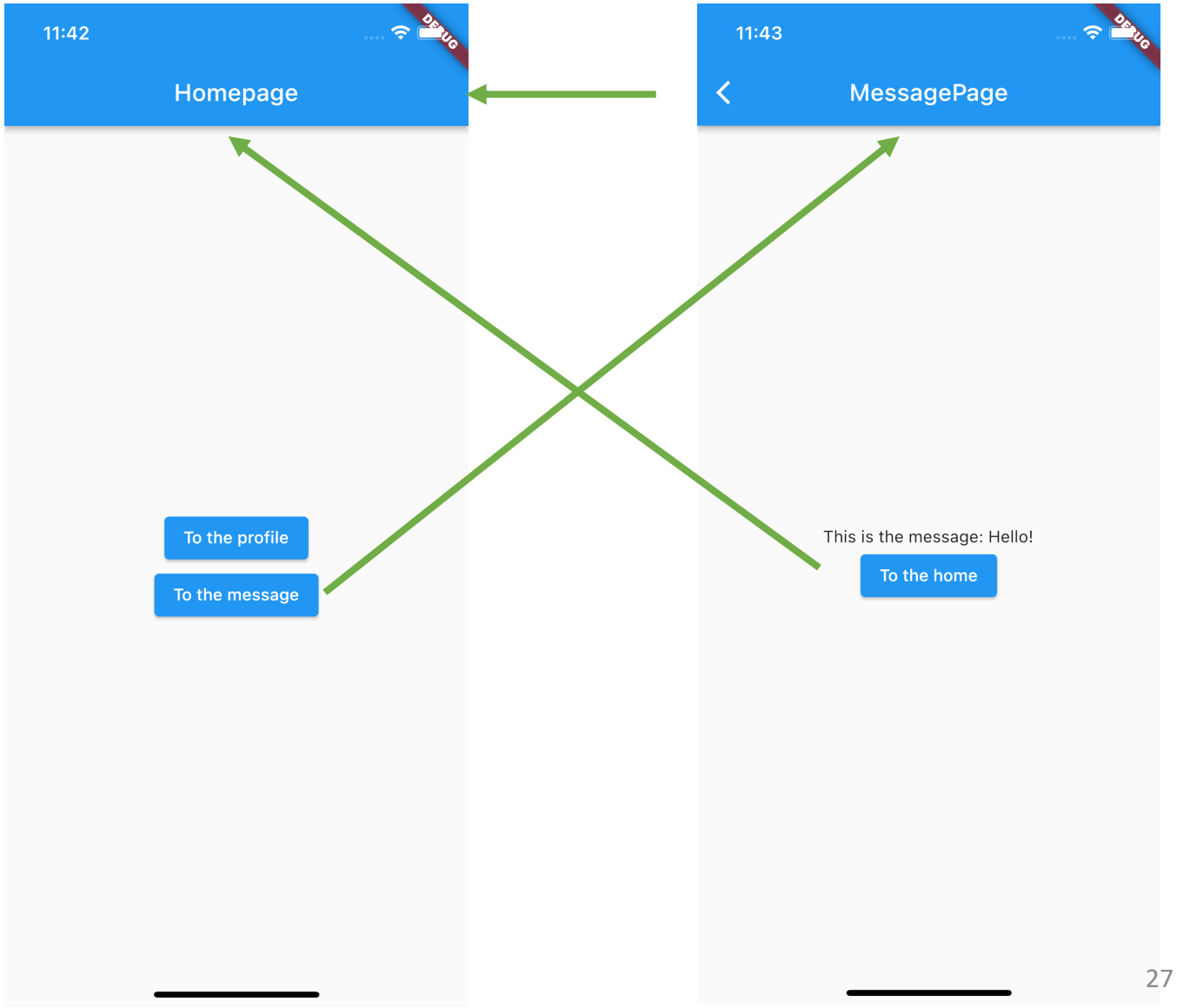
- Asynchrony
- Navigator
- Navigate to a new screen and back
- Named routes
- **Passing argument to a named routes**
- Returning an argument from a named route

- Exercise
- Homework
- Resources

Navigator – Passing an argument

- It is (of course) possible to pass arguments to the new route that can be used for several purposes.

- To demonstrate how, let's expand the app with another route `MessagePage` that will get an argument from the `HomePage` and will show it in the center of the screen.



Passing arguments – messagePage.dart boilerplate

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

class MessagePage extends StatelessWidget {

  const MessagePage({Key? key}) :
super(key: key);

  static const routename = 'MessagePage';

  @override

  Widget build(BuildContext context) {

    //TODO: get the message from HomePage

    return Scaffold(

      appBar: AppBar(

        title: Text(MessagePage.routename),

      ),

      ...
```

```
...
body: Center(
        child: Column(
          mainAxisAlignment:
MainAxisAlignment.center,
          children: [
            Text(''), //TODO: put the message
inside the Text here
            ElevatedButton(
              child: Text('To the home'),
              onPressed: () {
                Navigator.pop(context);
              },
            ),
          ],
        ),
      ),
    );
  } //build
} // MessagePage
```

Passing arguments – Add the new route and UI

- New route? Let's add it to the list:

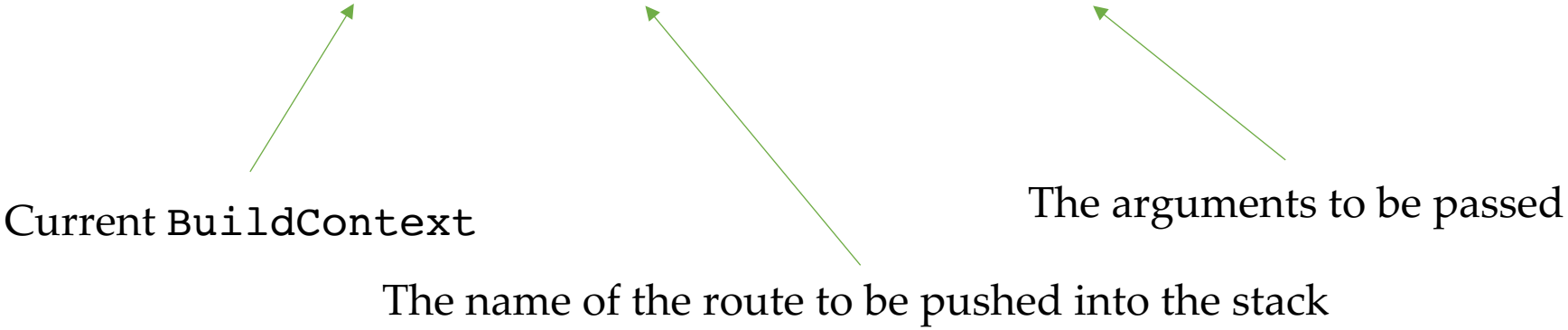
```
MaterialApp(
  initialRoute: '/',
  routes: {
    '/' : (context) => HomePage(),
    '/profile/': (context) => ProfilePage(),
    '/message/': (context) => MessagePage(),
  },
);
```

- To do: add a button in the `HomePage` to navigate to `MessagePage`

Passing arguments

- To pass an argument to the `MessagePage` route, now you can invoke `Navigator.pushNamed()` as:

```
...
onPressed: () {
  Navigator.pushNamed(context,'/message/', arguments: 'Hello!');
},
...
```



- Note that you can pass ANYTHING as argument, not just a `String`.

Retrieving arguments

- To retrieve the argument from the `MessagePage` route side you can use a `ModalRoute` as:

```
...
final message = ModalRoute.of(context)!.settings.arguments! as String;
...
```

To figure out what this is, you can imagine that as a utility that stands between the previous route (here `HomePage`) and the current one (here `MessagePage`). For details see: https://api.flutter.dev/flutter/widgets/ModalRoute-class.html

We put the `!` here to force the non-null type.

Arguments is an `Object?` But you know this is a `String`, so parse it explicitely!

- Then we display the retrieved argument by simply:

```
...
Text('This is the message: $message'),
...
```

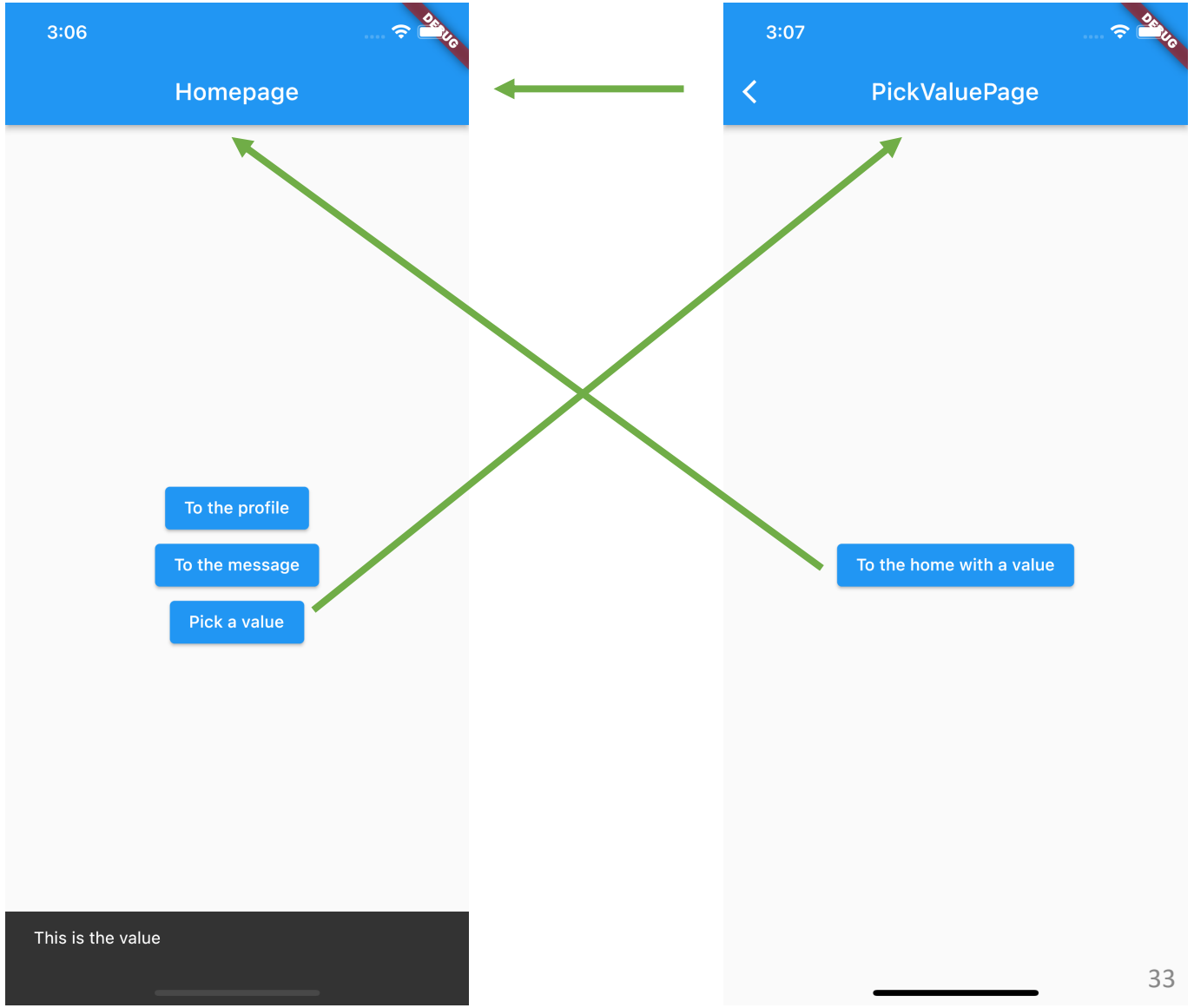
Outline

- Asynchrony
- Navigator
- Navigate to a new screen and back
- Named routes
- Passing argument to a named routes
- **Returning an argument from a named route**

- Exercise
- Homework
- Resources

Navigator – Returning data

- It is (of course) also possible to return data from a route.
- To demonstrate how, let's expand the app with another route `PickValuePage` that will provide a value to the `HomePage` which will be in charge of showing it via a `ScaffoldMessenger`.



Returning data – pickValuePage.dart boilerplate

```
import 'package:flutter/material.dart';
class PickValuePage extends StatelessWidget {
  const PickValuePage({Key? key}) :
super(key: key);
  static const routename = ‘PickValuePage';
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(PickValuePage.routename),
      ),
      ...
```

```
...
body: Center(
        child: ElevatedButton(
          child: Text('To the home'),
          onPressed: () {
            //TODO: implement the
navigation + return the data
          },
        ),
      ),
    );
  } //build
} //PickValuePage
```

Returning data – Add the new route

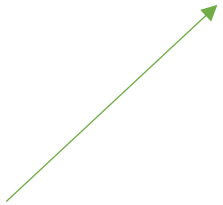
- New route? Let's add it to the list:

```
MaterialApp(
  initialRoute: '/',
  routes: {
    '/' : (context) => HomePage(),
    '/profile/': (context) => ProfilePage(),
    '/message/': (context) => MessagePage(),
    '/pickValue/': (context) => PickValuePage(),
  },
);
```

Returning arguments

- To return an argument to the `HomePage` route, you can invoke `Navigator.pop()` as:

```
...
onPressed: () {
  Navigator.pop(context, 'This is the value');
},
...
```



The value that will return to the `HomePage` once `PickValuePage` is popped out from the stack

- Note that you can return ANYTHING, not just a `String`.

Returning arguments

- To get the result, the `HomePage` must be patient and *await* for it::

Await means async stuff. The `onPressed` function become asynchronous as well so…

```
...
onPressed: () async {
  final result = await Navigator.pushNamed(context, '/pickValue/');
  ScaffoldMessenger.of(context)
    ..removeCurrentSnackBar()
    ..showSnackBar(SnackBar(content: Text('$result')));
},
...
```


Full example in lab_06-navigation/there_and_back_again/

Outline

- Asynchrony
- Navigator
- Navigate to a new screen and back
- Named routes
- Passing argument to a named routes
- Returning an argument from a named route

- **Exercise**
- Homework
- Resources

Exercises

- Exercise 06.01
  - Write an asynchronous function fetchUserRole() that after 3 seconds returns the String ‘admin’. Then, use that function in the main function to print the provided and properly produce the following output:

```
Fetching user role…
The user is an admin.
```

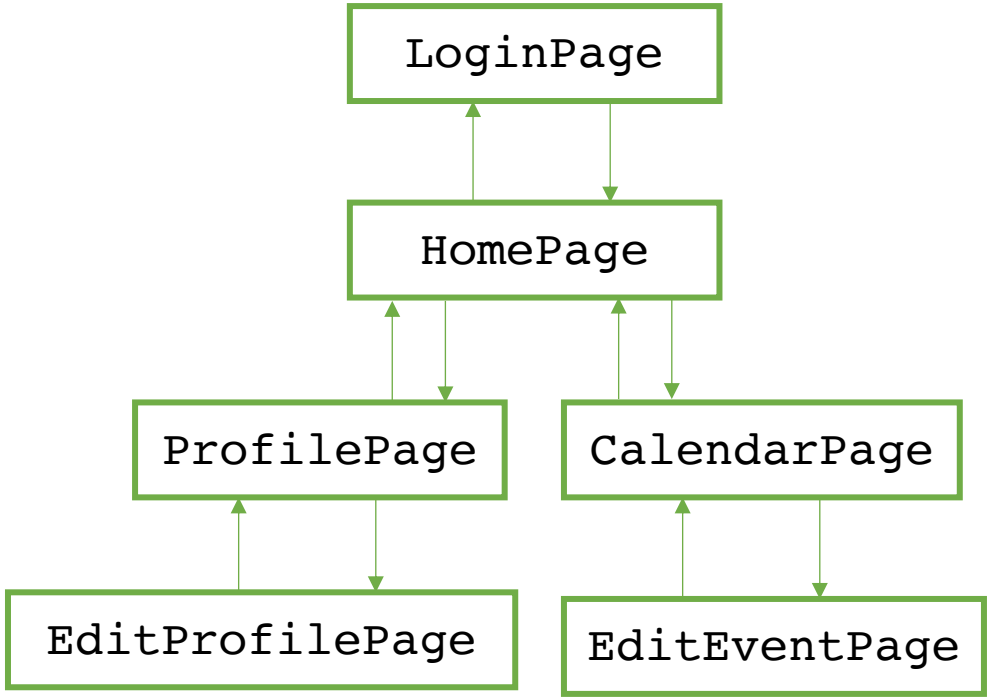
- Exercise 06.02
  - Use the fetchUserRole() function developed in 06.01 to create a new function isAdminUser() that checks if the string provided by fetchUserRole() is ‘admin’ and returns the respective boolean. Use the new function in the main to produce the following output:

```
Checking if user is an admin…
Ok, access granted! (if the user is an admin)
Access denied! (if the user is not an admin)
```

Exercise

- Exercise 06.03 (easy)

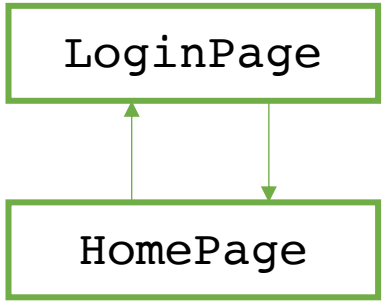
  - Create a new project ‘reproduce_structure’
  - Reproduce the app navigation structure on the right using the **named routing approach**.



Exercise

- Exercise 06.04 (medium)

  - Create a new project ‘login_flow’
  - Reproduce the app navigation structure on the right using the named routing approach.
  - The login page consists of a form with two textboxes (one for the username and the other for the password) and a button. Hint: you can use the widgets
  - When the user types “bug@expert.com” in the username textbox and “5TrNgP5Wd” in the password textbox, and taps the button, the user is redirected to the Homepage. If the credentials are wrong, a `ScaffoldMessenger` is showed for 2 seconds saying “Wrong credentials”.
  - The `HomePage` must show the provided username.

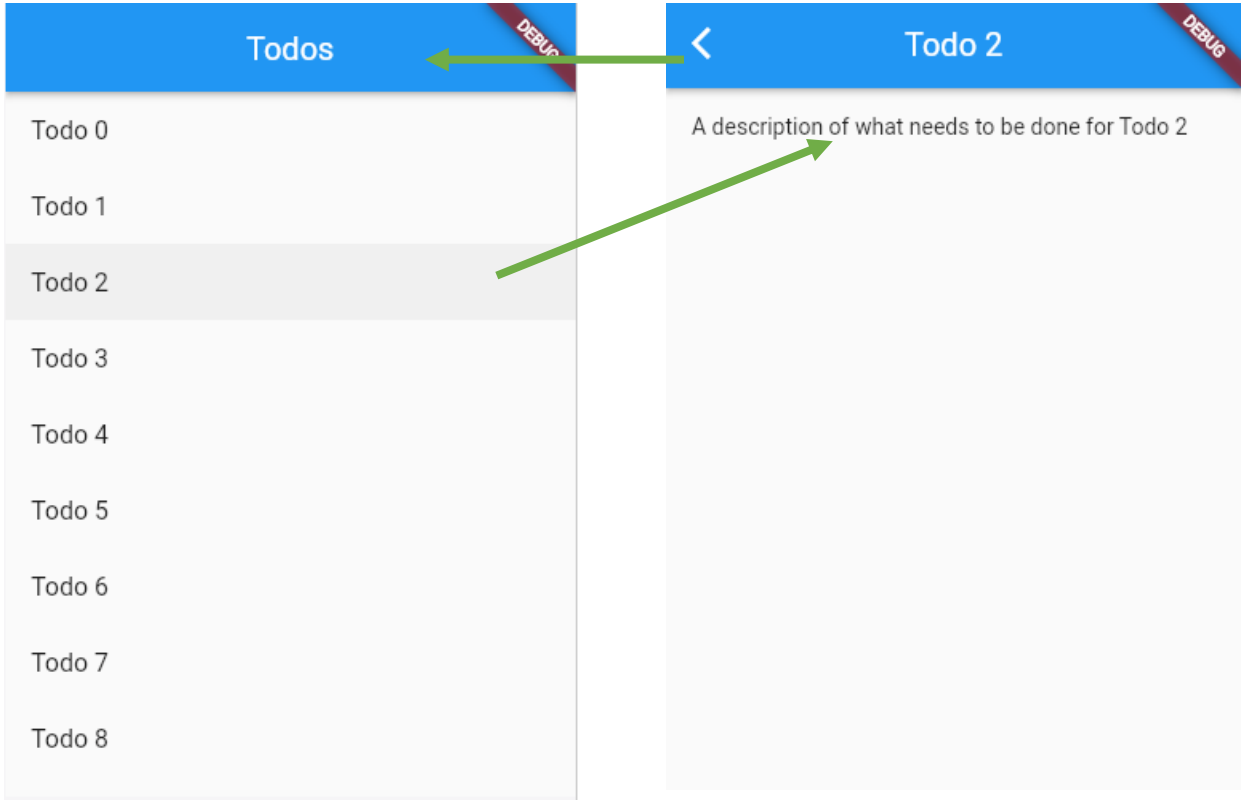


Exercise

- Exercise 06.05 (medium)

  - Follow the cookbook https://docs.flutter.dev/cookbook/navigation/passing-data by the Flutter team to learn how to pass data to a route directly to its constructor.

  - (solution available from the Flutter team in the cookbook)



Outline

- Recap
- Navigator
- Navigate to a new screen and back
- Named routes
- Passing argument to a named routes
- Returning an argument from a named route

- Exercise
- **Homework**
- Resources

Homework

- Get familiar with Asynchrony and Navigator

Outline

- Recap
- Navigator
- Navigate to a new screen and back
- Named routes
- Passing argument to a named routes
- Returning an argument from a named route

- Exercise
- Homework
- **Resources**

Resources

- Code repository of today's lesson and exercises solution
  - https://github.com/gcappon/bwthw/tree/master/lab_06-navigation
- Async and await codelabs
  - https://dart.dev/codelabs/async-await
- Navigation Recipes
  - https://docs.flutter.dev/cookbook/navigation