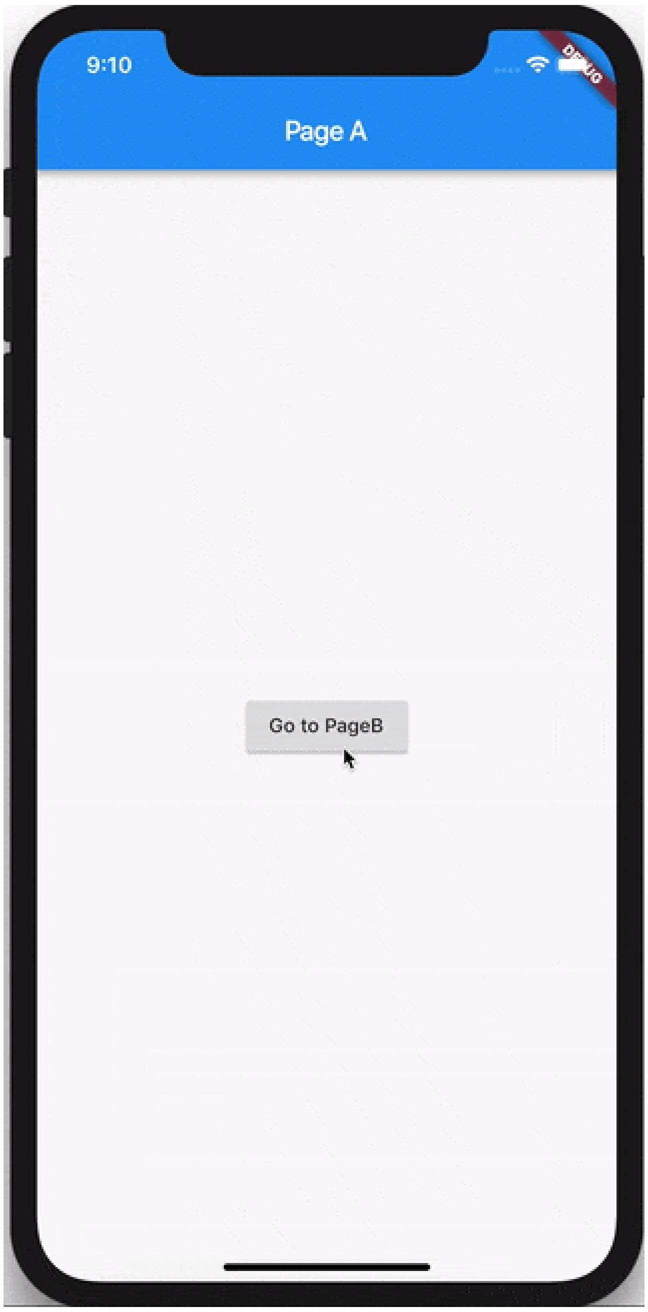
Recipes: Navigation

> In this recipe, we're going to take a look at how to use `BlocBuilder` and/or `BlocListener` to do navigation. We're going to explore two approaches: Direct Navigation and Route Navigation.

Direct Navigation

> In this example, we're going to take a look at how to use `BlocBuilder` to show a specific page (widget) in response to a state change in a bloc without the use of a route.



Bloc

Let's build `MyBloc` which will take `MyEvents` and convert them into `MyStates`.

MyEvent

For simplicity, our `MyBloc` will only respond to a two `MyEvents`: `eventA` and `eventB`.

```dart
enum MyEvent { eventA, eventB }
```

MyState

Our `MyBloc` can have one of two different `DataStates` :

- `StateA` - the state of the bloc when `PageA` is rendered.
- `StateB` - the state of the bloc when `PageB` is rendered.

```dart
abstract class MyState {}

class StateA extends MyState {}

class StateB extends MyState {}
```

MyBloc

Our `MyBloc` should look something like this:

```dart
import 'package:bloc/bloc.dart';

class MyBloc extends Bloc<MyEvent, MyState> {
  MyBloc() : super(StateA());

  @override
  Stream<MyState> mapEventToState(MyEvent event) async* {
    switch (event) {
      case MyEvent.eventA:
        yield StateA();
        break;
      case MyEvent.eventB:
        yield StateB();
        break;
    }
  }
}
```

UI Layer

Now let's take a look at how to hook up our `MyBloc` to a widget and show a different page based on the bloc state.

```dart
import 'package:flutter/material.dart';
import 'package:meta/meta.dart';
import 'package:bloc/bloc.dart';
import 'package:flutter_bloc/flutter_bloc.dart';

void main() {
  runApp(
    BlocProvider(
      create: (context) => MyBloc(),
      child: MyApp(),
    ),
  );
}

enum MyEvent { eventA, eventB }

@immutable
abstract class MyState {}

class StateA extends MyState {}

class StateB extends MyState {}

class MyBloc extends Bloc<MyEvent, MyState> {
  MyBloc() : super(StateA());

  @override
  Stream<MyState> mapEventToState(MyEvent event) async* {
    switch (event) {
      case MyEvent.eventA:
        yield StateA();
        break;
      case MyEvent.eventB:
        yield StateB();
        break;
    }
  }
}
```

```dart
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: BlocBuilder<MyBloc, MyState>(
        builder: (_, state) => state is StateA ? PageA() : PageB(),
      ),
    );
  }
}

class PageA extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Page A'),
      ),
      body: Center(
        child: ElevatedButton(
          child: Text('Go to PageB'),
          onPressed: () {
            BlocProvider.of<MyBloc>(context).add(MyEvent.eventB);
          },
        ),
      ),
    );
  }
}

class PageB extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Page B'),
      ),
      body: Center(
        child: ElevatedButton(
          child: Text('Go to PageA'),
          onPressed: () {
            BlocProvider.of<MyBloc>(context).add(MyEvent.eventA);
          },
          ),
        ),
      );
    }
  }
```

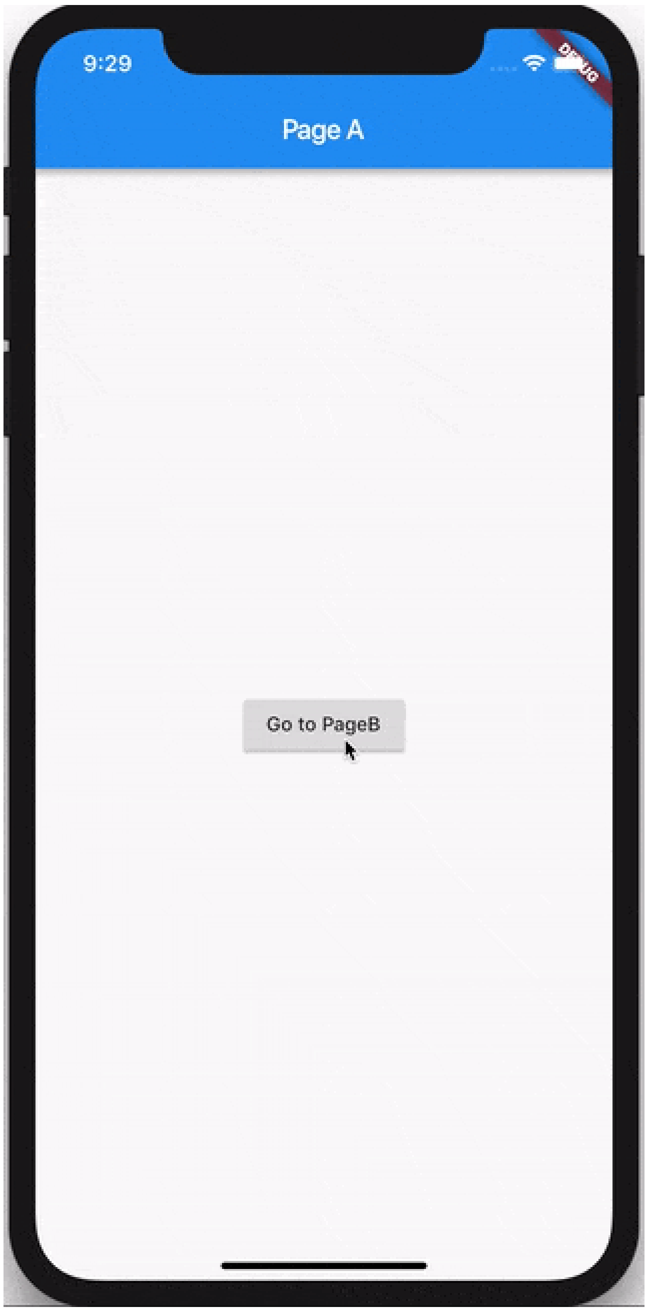
We use the `BlocBuilder` widget in order to render the correct widget in response to state changes in our `MyBloc` .

We use the `BlocProvider` widget in order to make our instance of `MyBloc` available to the entire widget tree.

The full source for this recipe can be found here.

Route Navigation

> In this example, we're going to take a look at how to use `BlocListener` to navigate to a specific page (widget) in response to a state change in a bloc using a route.



Bloc

We're going to reuse the same `MyBloc` from the previous example.

UI Layer

Let's take a look at how to route to a different page based on the state of `MyBloc`.

```dart
import 'package:flutter/material.dart';
import 'package:meta/meta.dart';
import 'package:bloc/bloc.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
```

```dart
void main() {
  runApp(
    BlocProvider(
      create: (context) => MyBloc(),
      child: MyApp(),
    ),
  );
}

enum MyEvent { eventA, eventB }

@immutable
abstract class MyState {}

class StateA extends MyState {}

class StateB extends MyState {}

class MyBloc extends Bloc<MyEvent, MyState> {
  MyBloc() : super(StateA());

  @override
  Stream<MyState> mapEventToState(MyEvent event) async* {
    switch (event) {
      case MyEvent.eventA:
        yield StateA();
        break;
      case MyEvent.eventB:
        yield StateB();
        break;
    }
  }
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      routes: {
        '/': (context) => PageA(),
        '/pageB': (context) => PageB(),
      },
      initialRoute: '/',
    );
  }
}

class PageA extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return BlocListener<MyBloc, MyState>(
      listener: (context, state) {
        if (state is StateB) {
          Navigator.of(context).pushNamed('/pageB');
        }
      },
      child: Scaffold(
        appBar: AppBar(
          title: Text('Page A'),
        ),
        body: Center(
          child: ElevatedButton(
            child: Text('Go to PageB'),
            onPressed: () {
              BlocProvider.of<MyBloc>(context).add(MyEvent.eventB);
            },
          ),
        ),
      ),
    );
  }
}

class PageB extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Page B'),
      ),
      body: Center(
        child: ElevatedButton(
          child: Text('Pop'),
          onPressed: () {
            Navigator.of(context).pop();
          },
        ),
      ),
    );
  }
}
```

We use the `BlocListener` widget in order to push a new route in response to state changes in our `MyBloc`.

! For the sake of this example we are adding an event just for navigation. In a real application, you should not create explicit navigation events. If there is no "business logic" necessary in order to trigger navigation you should always directly navigate in response to user input (in the `onPressed` callback, etc...). Only navigate in response to state changes if some "business logic" is required in order to determine where to navigate.

The full source for this recipe can be found here.