Flutter Navigation:

Animate a Widget across the screen.

- It’s often helpful to guide users through an app as they navigate from screen to screen. A common technique to lead users through an app is to animate a widget from one screen to the next. This creates a visual anchor connecting the two screens.

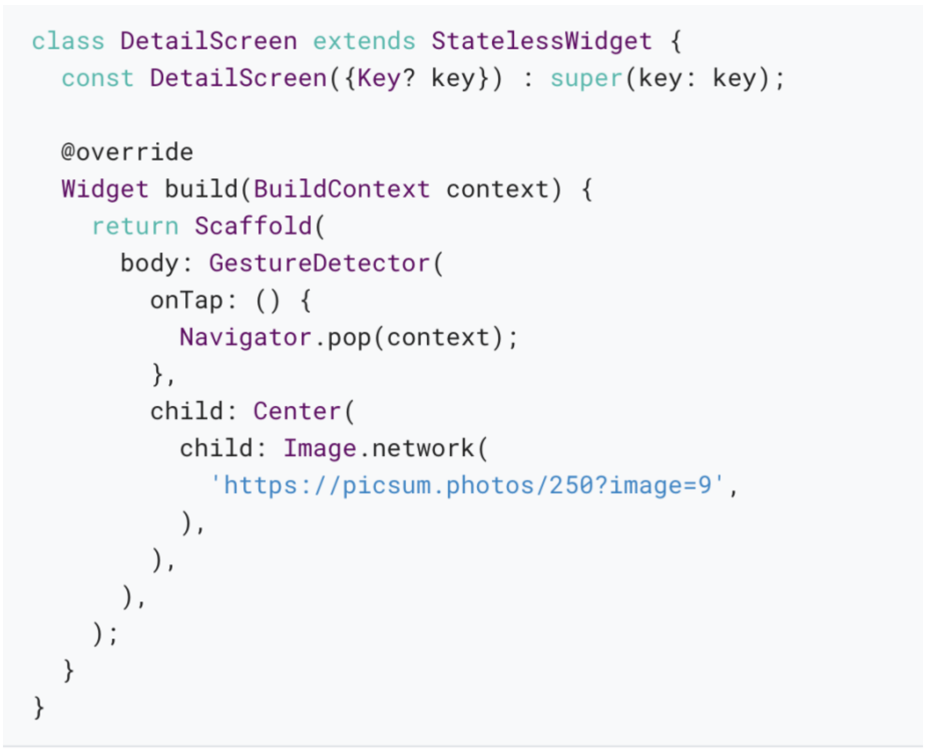
Use the Hero widget to animate a widget from one screen to the next. This recipe uses the following steps:

1. Create two screens showing the same image.
2. Add a Hero widget to the first screen.
3. Add a Hero widget to the second screen.

Create two screens showing the same image.

- In this example, display the same image on both screens. Animate the image from the first screen to the second screen when the user taps the image. For now, create the visual structure; handle animations in the next steps.

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

class MainScreen extends StatelessWidget {
  const MainScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Main Screen'),
      ),
      body: GestureDetector(
        onTap: () {
          Navigator.push(context, MaterialPageRoute(builder: (context) {
            return const DetailScreen();
          }));
        },
        child: Image.network(
          'https://picsum.photos/250?image=9',
        ),
      ),
    );
  }
}
```



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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: GestureDetector(
        onTap: () {
          Navigator.pop(context);
        },
        child: Center(
          child: Image.network(
            'https://picsum.photos/250?image=9',
          ),
        ),
      ),
    );
  }
}
```

Add Hero Widget to the first screen.

To connect the two screens together with an animation, wrap the Image widget on both screens in a Hero widget. The Hero widget requires two arguments:

`tag`

An object that identifies the `Hero`. It must be the same on both screens.

`child`

The widget to animate across screens.

```dart
Hero(
  tag: 'imageHero',
  child: Image.network(
    'https://picsum.photos/250?image=9',
  ),
)
```

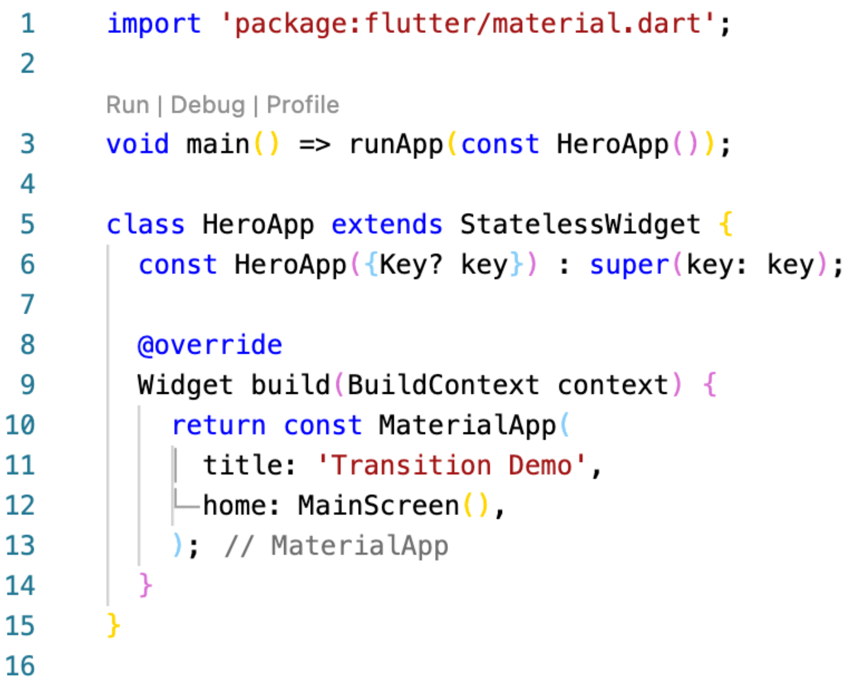
Add a Hero widget to the second screen

To complete the connection with the first screen, wrap the Image on the second screen with a Hero widget that has the same tag as the Hero in the first screen.

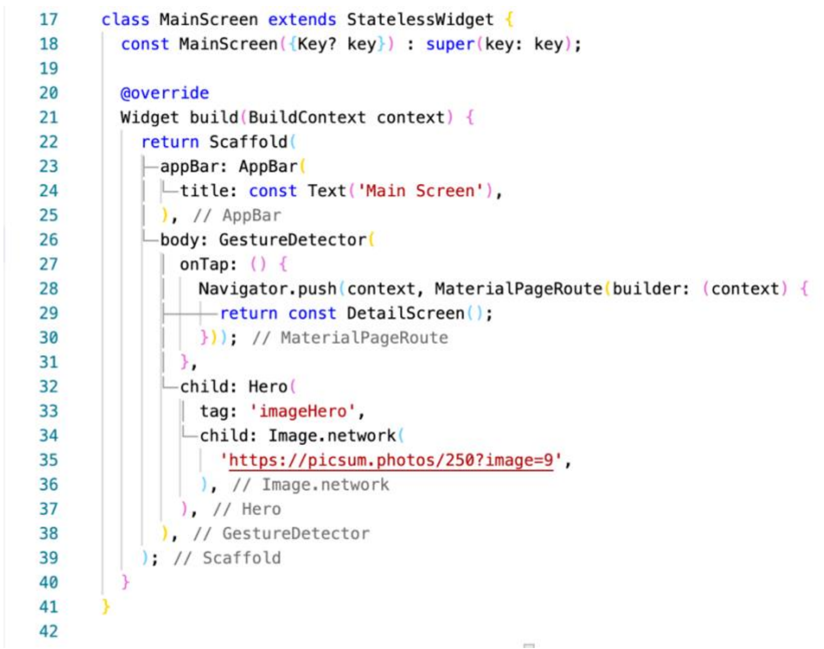
After applying the Hero widget to the second screen, the animation between screens just works.

```
Hero(
  tag: 'imageHero',
  child: Image.network(
    'https://picsum.photos/250?image=9',
  ),
)
```

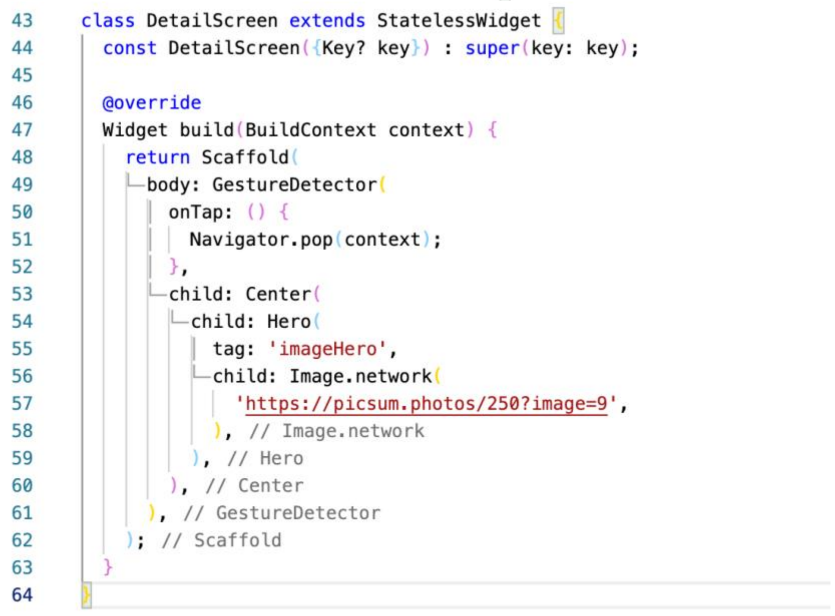
Code:



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

void main() => runApp(const HeroApp());

class HeroApp extends StatelessWidget {
  const HeroApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: 'Transition Demo',
      home: MainScreen(),
    ); // MaterialApp
  }
}

```



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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Main Screen'),
      ), // AppBar
      body: GestureDetector(
        onTap: () {
          Navigator.push(context, MaterialPageRoute(builder: (context) {
            return const DetailScreen();
          })); // MaterialPageRoute
        },
        child: Hero(
          tag: 'imageHero',
          child: Image.network(
            'https://picsum.photos/250?image=9',
          ), // Image.network
        ), // Hero
      ), // GestureDetector
    ); // Scaffold
  }
}

```



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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: GestureDetector(
        onTap: () {
          Navigator.pop(context);
        },
        child: Center(
          child: Hero(
            tag: 'imageHero',
            child: Image.network(
              'https://picsum.photos/250?image=9',
            ), // Image.network
          ), // Hero
        ), // Center
      ), // GestureDetector
    ); // Scaffold
  }
}
```

Navigate to new Screens 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.

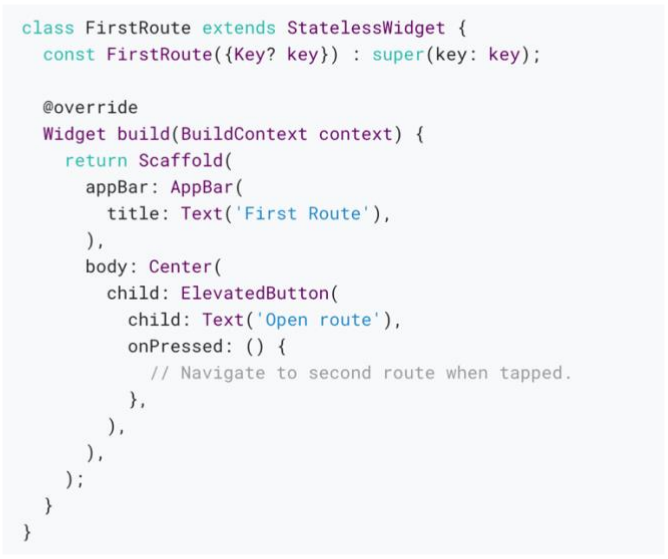
This recipe uses the Navigator to navigate to a new route.

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

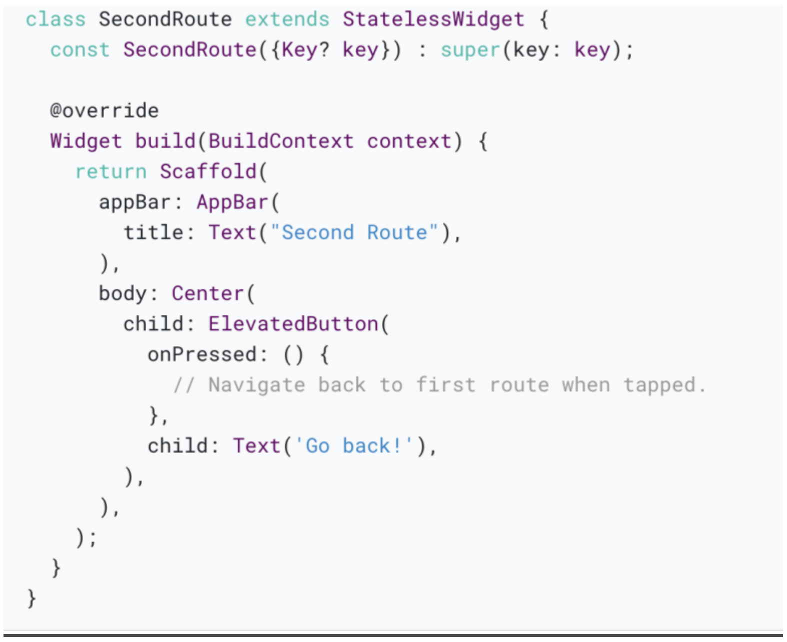
1. Create two routes.
2. Navigate to the second route using Navigator.push().
3. Return to the first route using Navigator.pop().

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.



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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('First Route'),
      ),
      body: Center(
        child: ElevatedButton(
          child: Text('Open route'),
          onPressed: () {
            // Navigate to second route when tapped.
          },
        ),
      ),
    );
  }
}
```



```
class SecondRoute extends StatelessWidget {
  const SecondRoute({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Second Route"),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            // Navigate back to first route when tapped.
          },
          child: Text('Go back!'),
        ),
      ),
    );
  }
}
```

<u>**Navigate to the Second route using Navigator.push() :**</u>

To switch to a new route, use the Navigator.push() method. The push() method adds a Route to the stack of routes managed by the Navigator. Where does the Route come from? You can create your own, or use a MaterialPageRoute, which is useful because it transitions to the new route using a platform-specific animation.

In the build() method of the FirstRoute widget, update the onPressed() callback:

```
// Within the `FirstRoute` widget
onPressed: () {
  Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => SecondRoute()),
  );
}
```

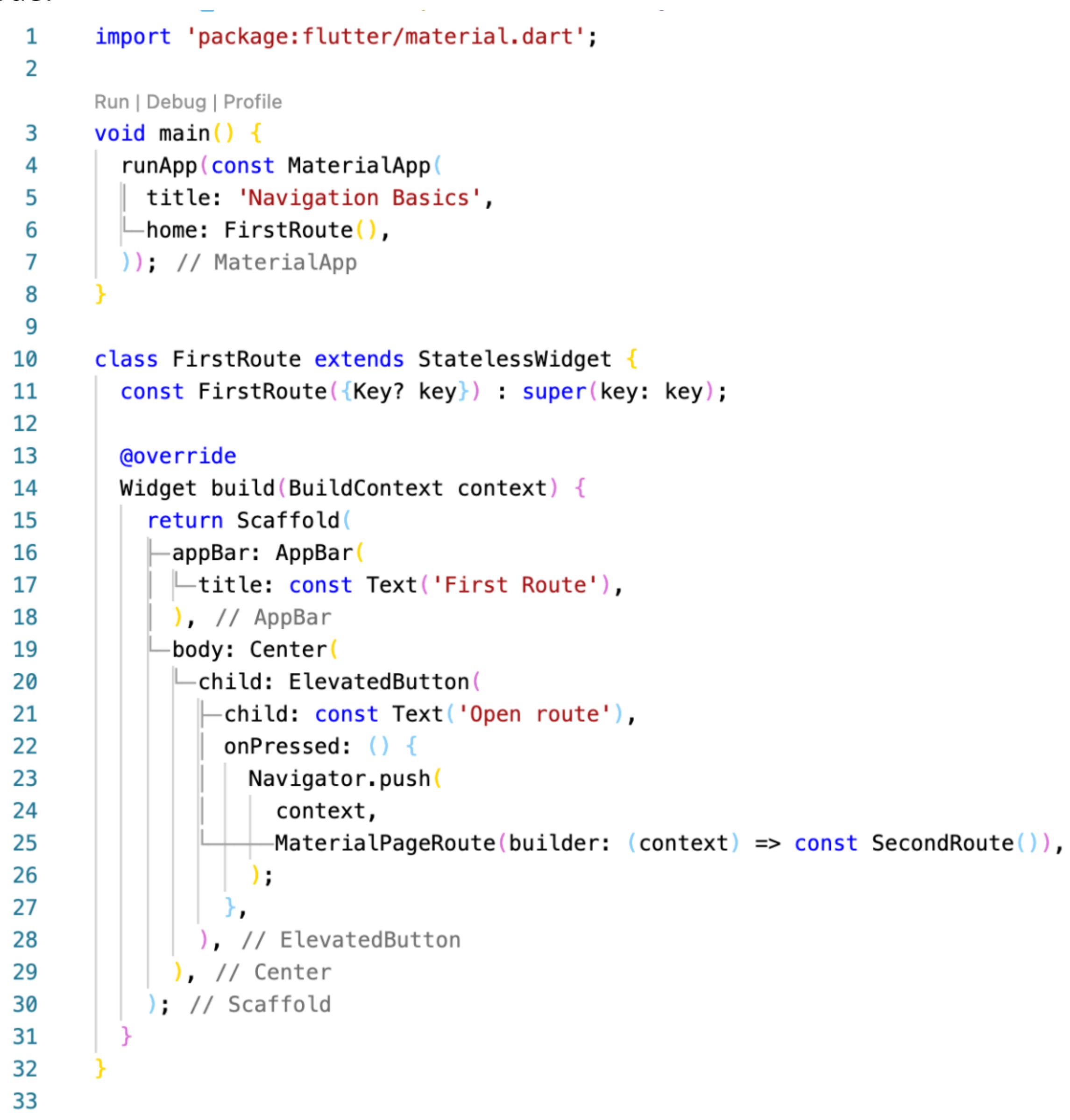
<u>**Return to the First route using Navigator.pop() :**</u>

How do you close the second route and return to the first? By using the Navigator.pop() method. The pop() method removes the current Route from the stack of routes managed by the Navigator.

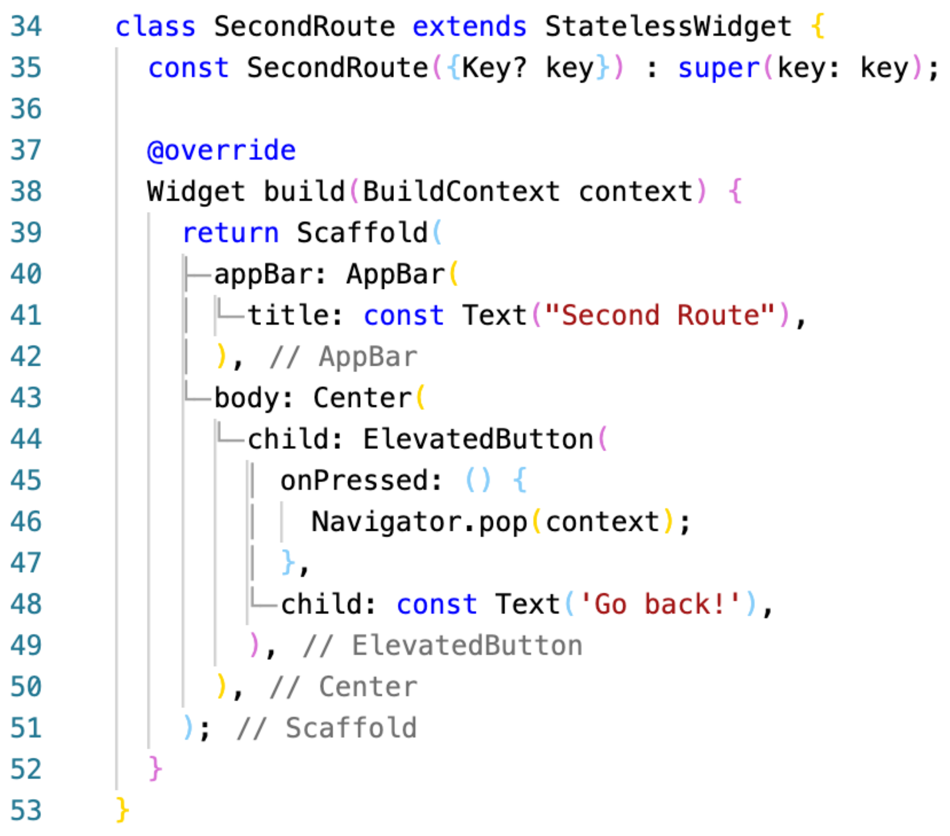
To implement a return to the original route, update the onPressed() callback in the SecondRoute widget:

```
// Within the SecondRoute widget
onPressed: () {
  Navigator.pop(context);
}
```

Code:



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

void main() {
  runApp(const MaterialApp(
    title: 'Navigation Basics',
    home: FirstRoute(),
  )); // MaterialApp
}

class FirstRoute extends StatelessWidget {
  const FirstRoute({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('First Route'),
      ), // AppBar
      body: Center(
        child: ElevatedButton(
          child: const Text('Open route'),
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => const SecondRoute()),
            );
          },
        ), // ElevatedButton
      ), // Center
    ); // Scaffold
  }
}

```



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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Second Route"),
      ), // AppBar
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.pop(context);
          },
          child: const Text('Go back!'),
        ), // ElevatedButton
      ), // Center
    ); // Scaffold
  }
}
```

Navigate with named routes:

In the Navigate to a new screen and back recipe, you learned how to navigate to a new screen by creating a new route and pushing it to the Navigator.

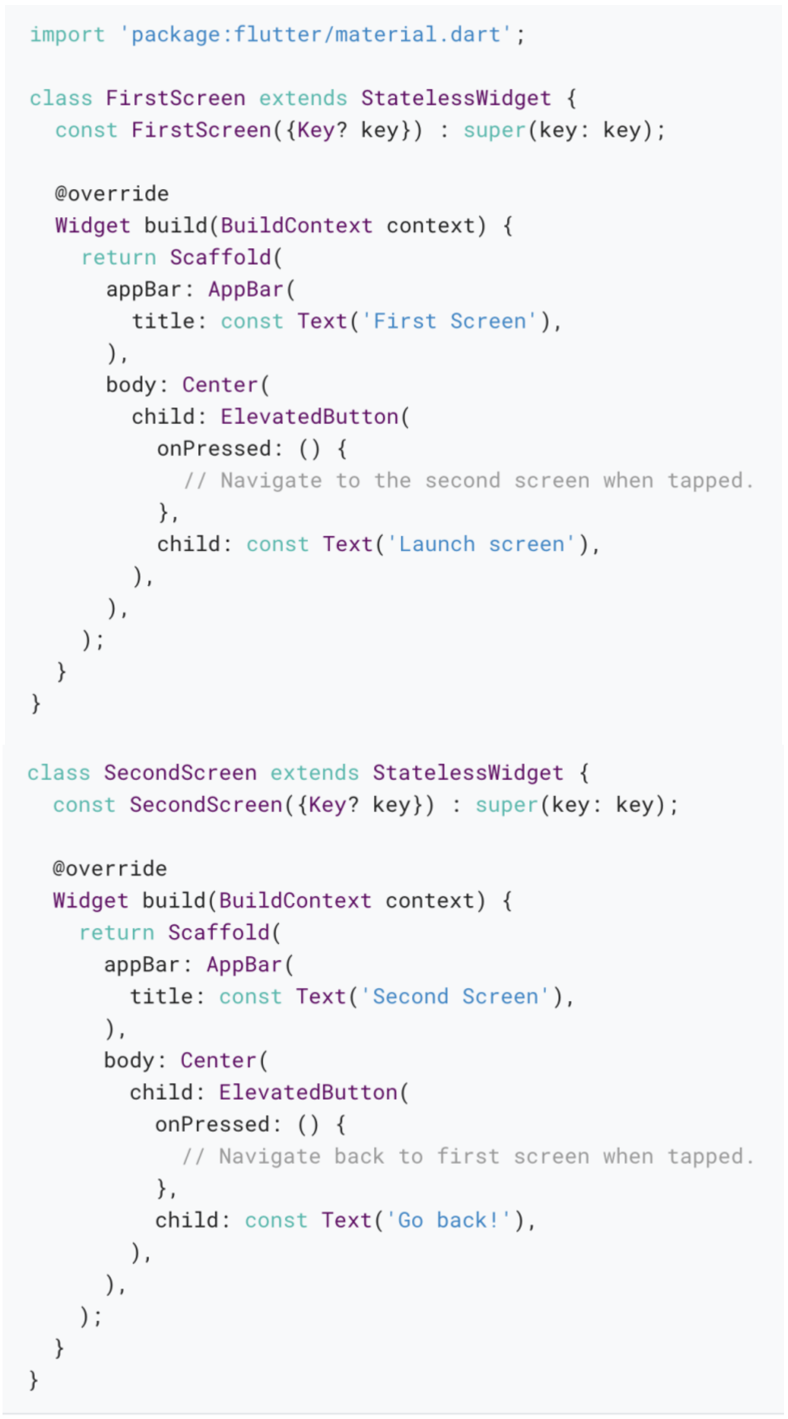
However, if you need to navigate to the same screen in many parts of your app, this approach can result in code duplication. The solution is to define a *named route*, and use the named route for navigation.

To work with named routes, use the Navigator.pushNamed() function. This example replicates the functionality from the original recipe, demonstrating how to use named routes using the following steps:

1. Create two screens.
2. Define the routes.
3. Navigate to the second screen using Navigator.pushNamed().
4. Return to the first screen using Navigator.pop().

1. Create two screens:

First, create two screens to work with. The first screen contains a button that navigates to the second screen. The second screen contains a button that navigates back to the first.



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

class FirstScreen extends StatelessWidget {
  const FirstScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('First Screen'),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            // Navigate to the second screen when tapped.
          },
          child: const Text('Launch screen'),
        ),
      ),
    );
  }
}

class SecondScreen extends StatelessWidget {
  const SecondScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Second Screen'),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            // Navigate back to first screen when tapped.
          },
          child: const Text('Go back!'),
        ),
      ),
    );
  }
}
```

2. Defines the routes:

Next, define the routes by providing additional properties to the MaterialApp constructor: the initialRoute and the routes themselves.

The initialRoute property defines which route the app should start with. The routes property defines the available named routes and the widgets to build when navigating to those routes.

```
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) => const FirstScreen(),
    // When navigating to the "/second" route, build the SecondScreen widget.
    '/second': (context) => const SecondScreen(),
  },
)
```

3. Navigate to the second screen using Navigator.pushNamed().

With the widgets and routes in place, trigger navigation by using the Navigator.pushNamed() method. This tells Flutter to build the widget defined in the routes table and launch the screen.

In the build() method of the FirstScreen widget, update the onPressed() callback:

```
// Within the `FirstScreen` widget
onPressed: () {
  // Navigate to the second screen using a named route.
  Navigator.pushNamed(context, '/second');
}
```

4. Return to the First Screen:

To navigate back to the first screen, use the Navigator.pop() function.

```
// Within the SecondScreen widget
onPressed: () {
  // Navigate back to the first screen by popping the current route
  // off the stack.
  Navigator.pop(context);
}
```