
Flutter- Workshop

Julian Hartl



Overview

Why Flutter?

Flutter Basics

Dart Crash-Course

Developing a Shopping List app

Building your own app

Outlook



Why Flutter?

What even is Flutter?

What even is Flutter?

“Flutter is an open source framework by Google for building beautiful, natively compiled, multi-platform applications from a single codebase.”

- Open source UI software development toolkit
- Supports many platforms
- Single codebase



Why Flutter?

**What is your experience with app development,
and what are your expectations for the workshop?**

Why Flutter?

What does Flutter offer?

What does Flutter offer?

- Cross-Platform development (Android, iOS, Web, Windows, macOS, Linux, Fuchsia)
- Performant due to Ahead-of-Time (AOT) Compilation
- Expressive and flexible UI (large set of prebuilt components)
- Hot reload (instant UI updates)
- Consistent behavior across all platforms
- Strong community and ecosystem (wide adoption in the industry)

Why Flutter?

Applications in the real world

17:37

iX xDrive50

Check Status

ALL
GOOD



Updating Vehicle Status...

Fully Charged

100 % / 504 km



Vehicle Finder

Zeppelinstraße 6, 85748
Garching bei München



Remote Cameras

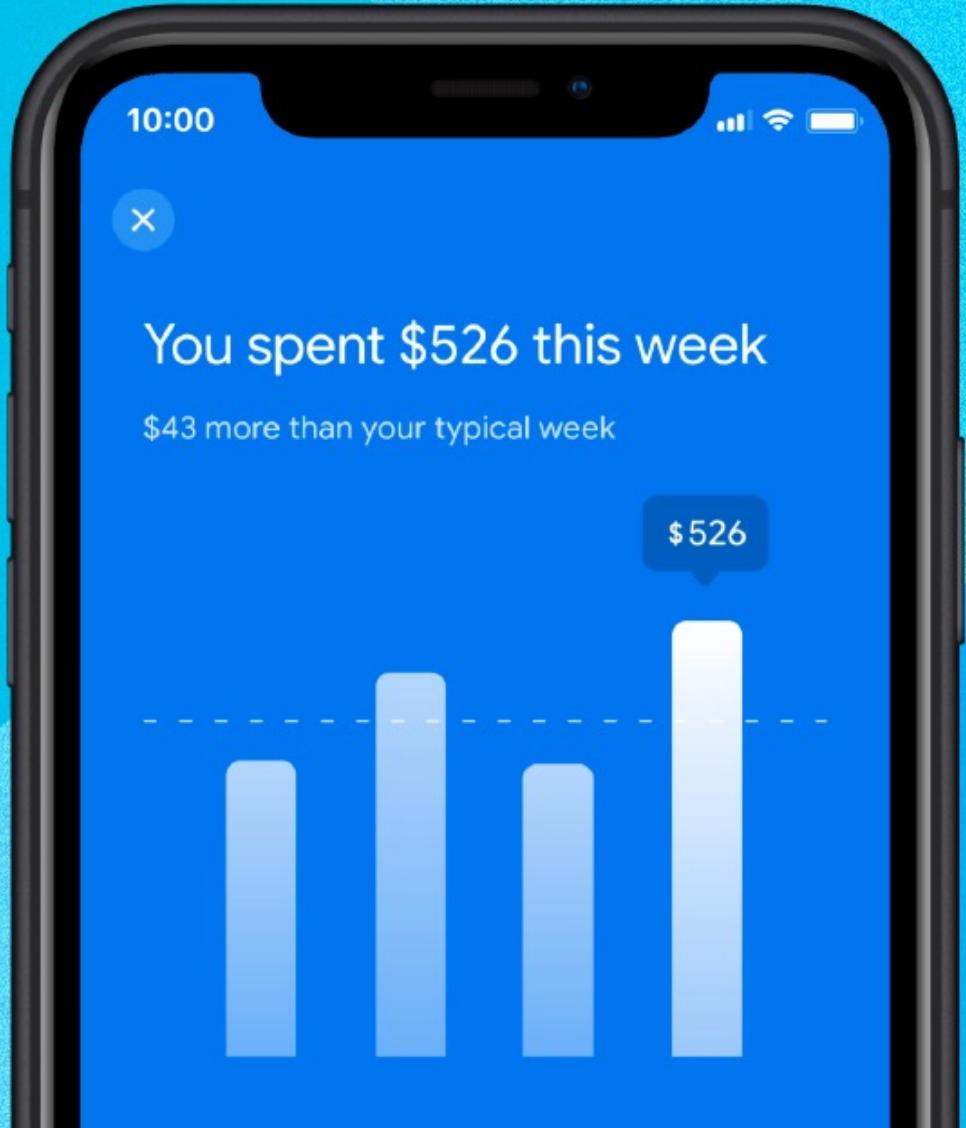
Remote 3D und Remote
Inside View



BMW

Feature and design discrepancies between their iOS and Android offering had grown too large

- Not yet mature enough
- Not web based => better UX
- Build & design once



10:00

X

You spent \$526 this week

\$43 more than your typical week

\$526

A smartphone screen showing a weekly spending summary. The time is 10:00. A blue 'X' button is in the top-left corner. The main message is "You spent \$526 this week" with a note "\$43 more than your typical week". A bar chart shows four bars of increasing height, with the tallest bar reaching \$526. A speech bubble above the bars says "\$526". The background is light blue.

Google-Pay

Building every feature twice

Cloud Heavy upfront investment of engineering resources (retraining, reapproval, ...)

- ✓ 70% reduction in engineering effort
- ✓ 35% reduction in lines of code
- ✓ Saved about 60-70% of their engineer's time



Search Cars & Trucks

eBay

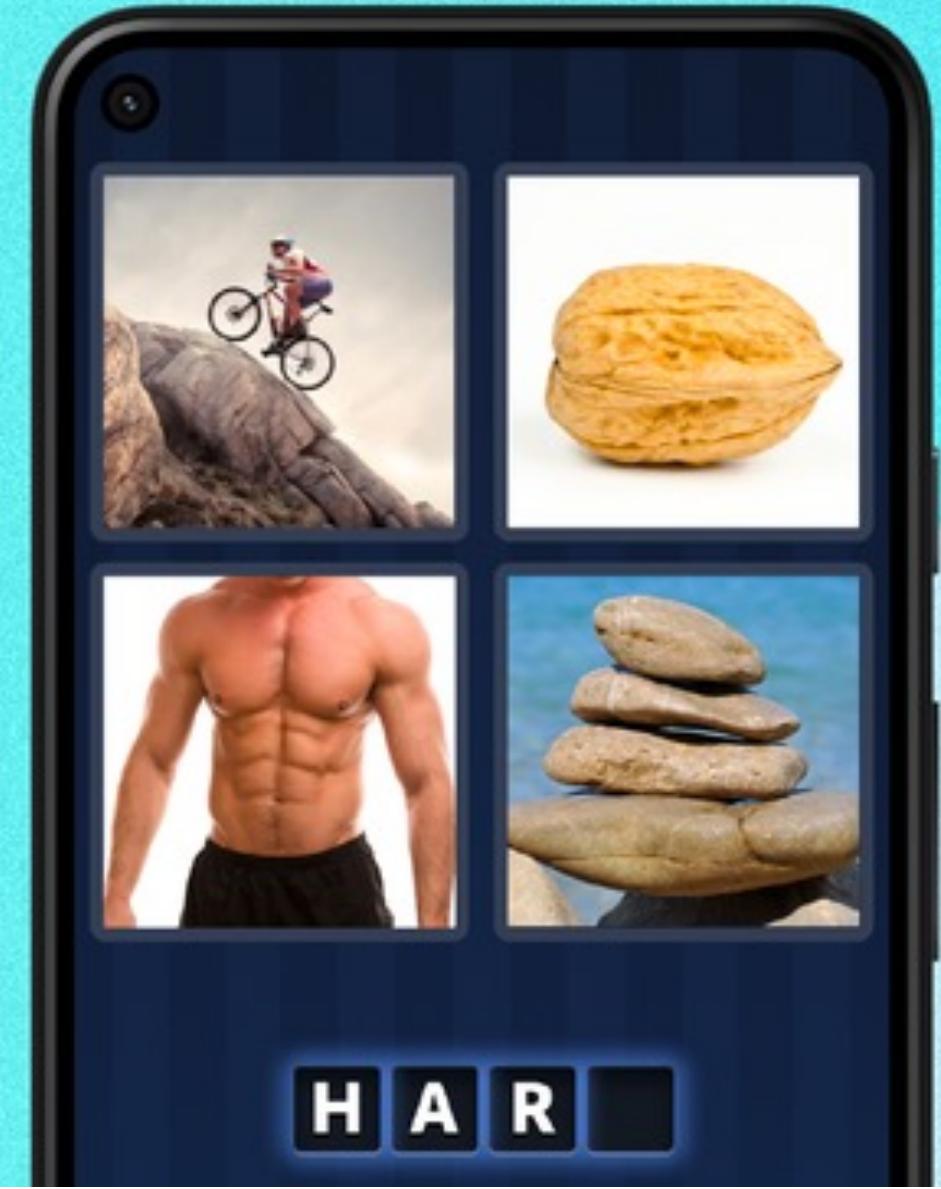
Building a new app to buy and sell hard-to-find cars for both Android and iOS in under a year

Time-saving and design capabilities

Hot reload

Built-in testing

Code sharing



4 Pics 1 Word

Giving 4 Pics 1 Word a tuneup

- No more troubleshooting cross-platform functionality
- Performs offline on low-end devices
- 99.9% crash-free rate

Why Flutter?

Flutter compared to React Native & Native



Flutter vs. React Native & Native

Flutter

- Cross-Platform
- Comprehensive set of prebuilt UI components following Material Design & Cupertino guidelines
- AOT compiled to native code, ensuring faster startup and performance
- Although younger, it has a rapidly growing community

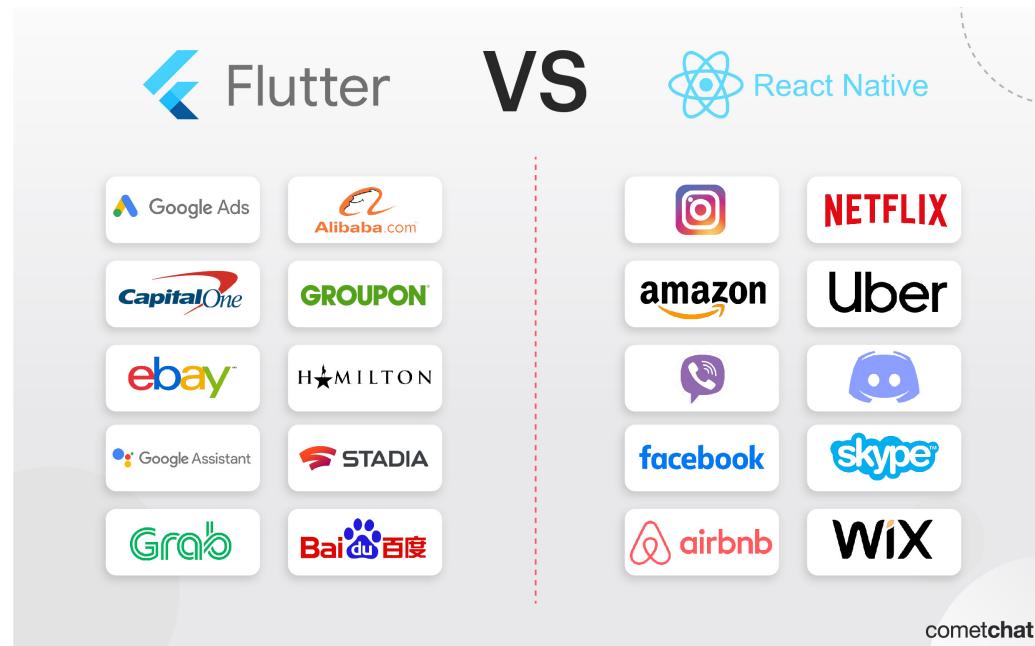
React Native

- Cross-Platform
- Core components, but for more complete UI, developers often rely on third-party libraries
- Uses JavaScript, which is interpreted at runtime and therefore only close to native performance
- Established community with a vast number of libraries

Native

- Only one platform
- Wealth of UI components available through the platform's SDKs
- Highest performance possible on their platform due to direct access to hardware and APIs
- Comprehensive tools and APIs covering every aspect of the respective platform

Flutter vs. React Native



Summary

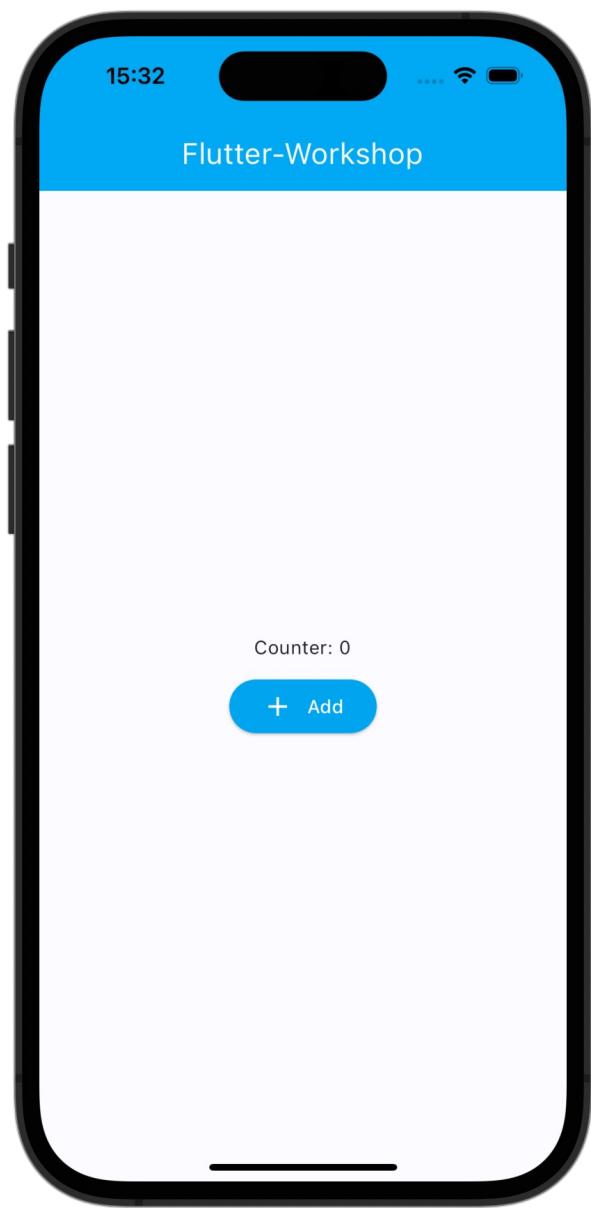


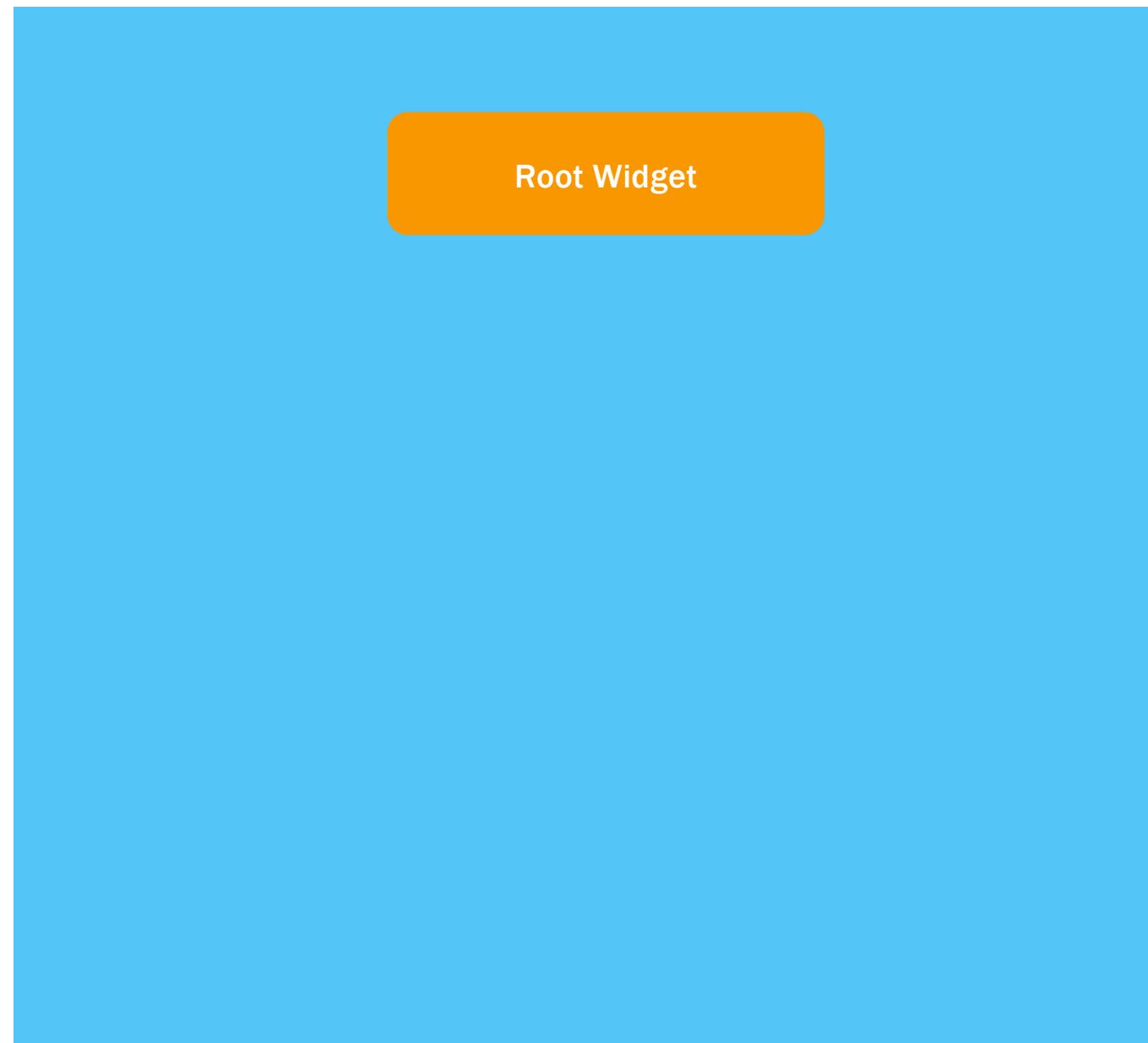
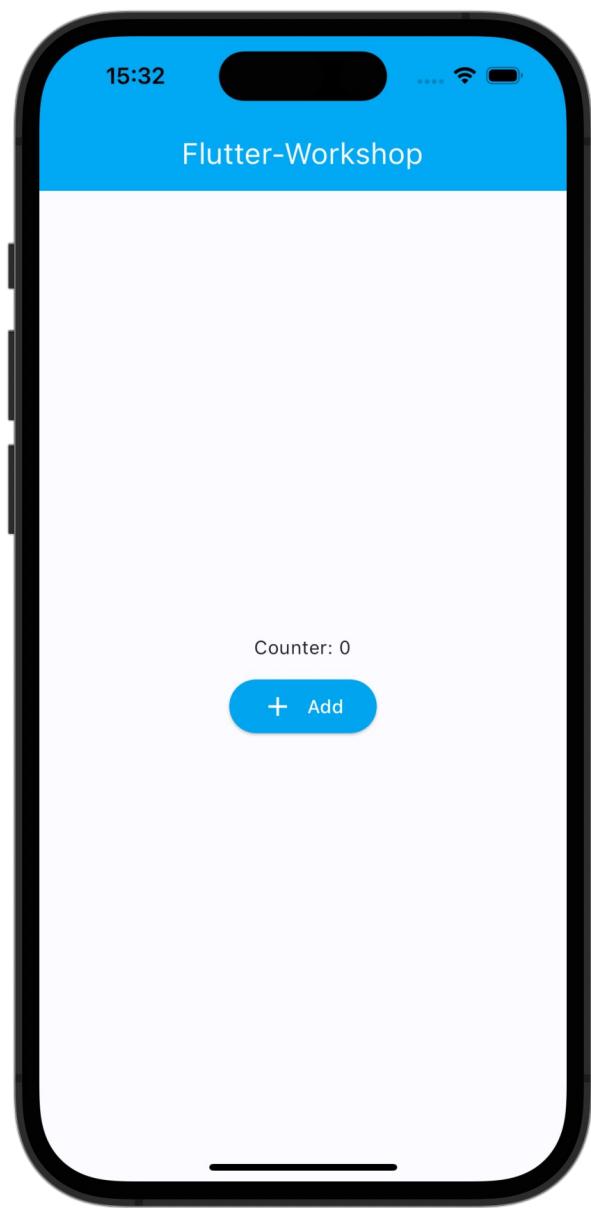
- Build once for all platforms
- Native performance
- Rich ecosystem & wide adoption

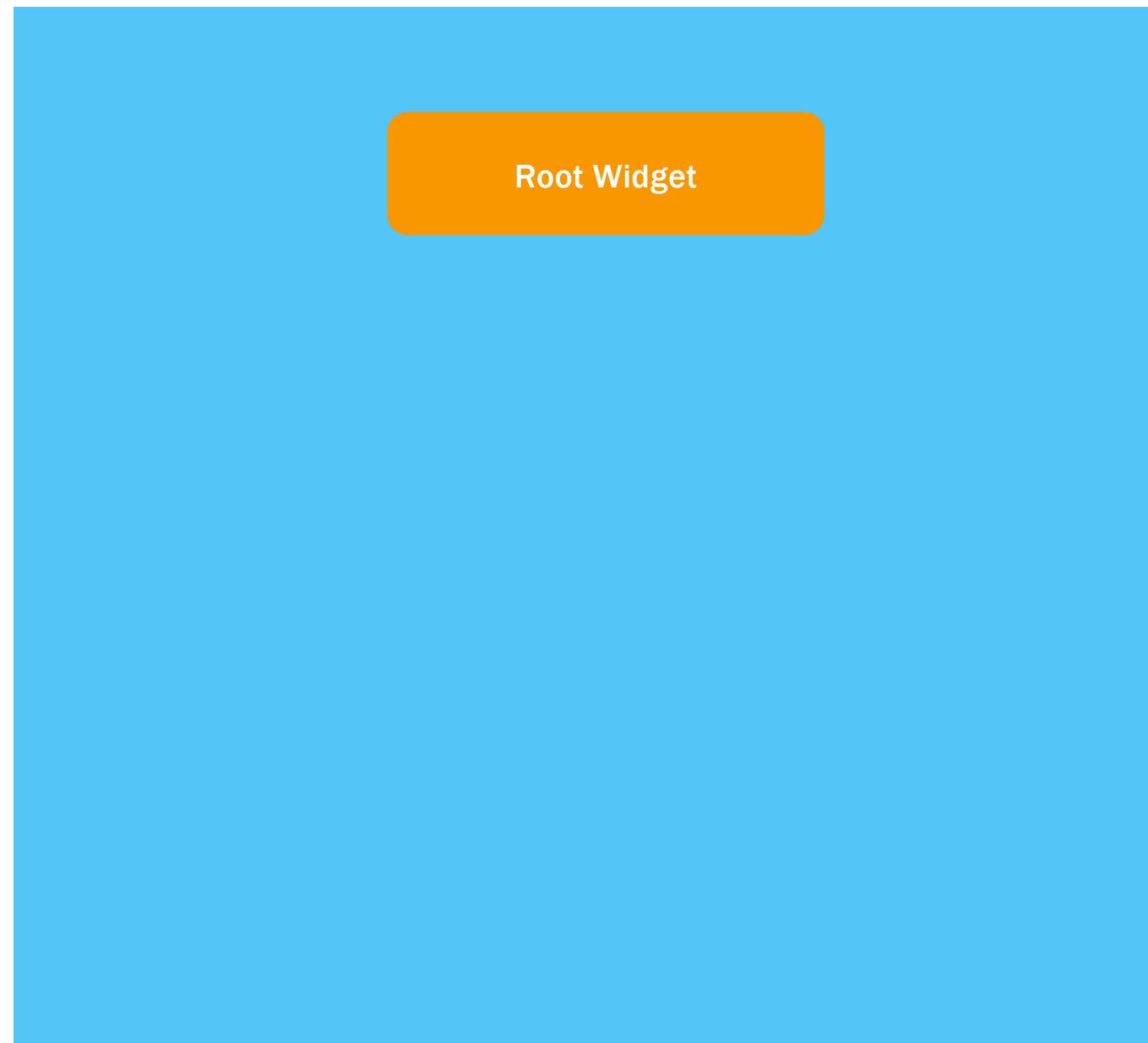
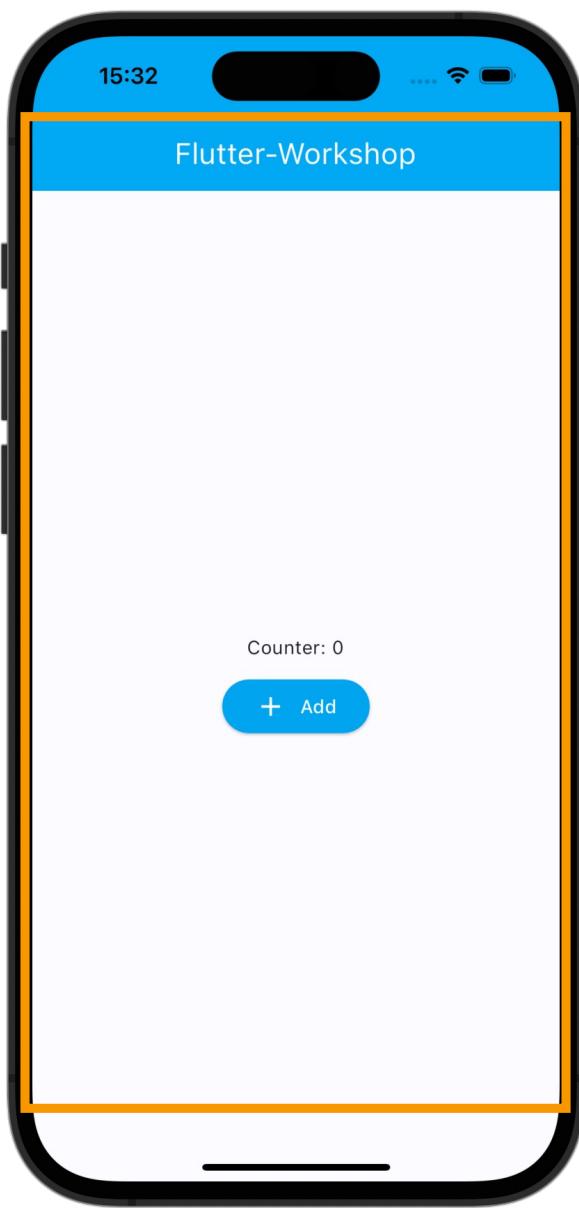
Flutter Basics

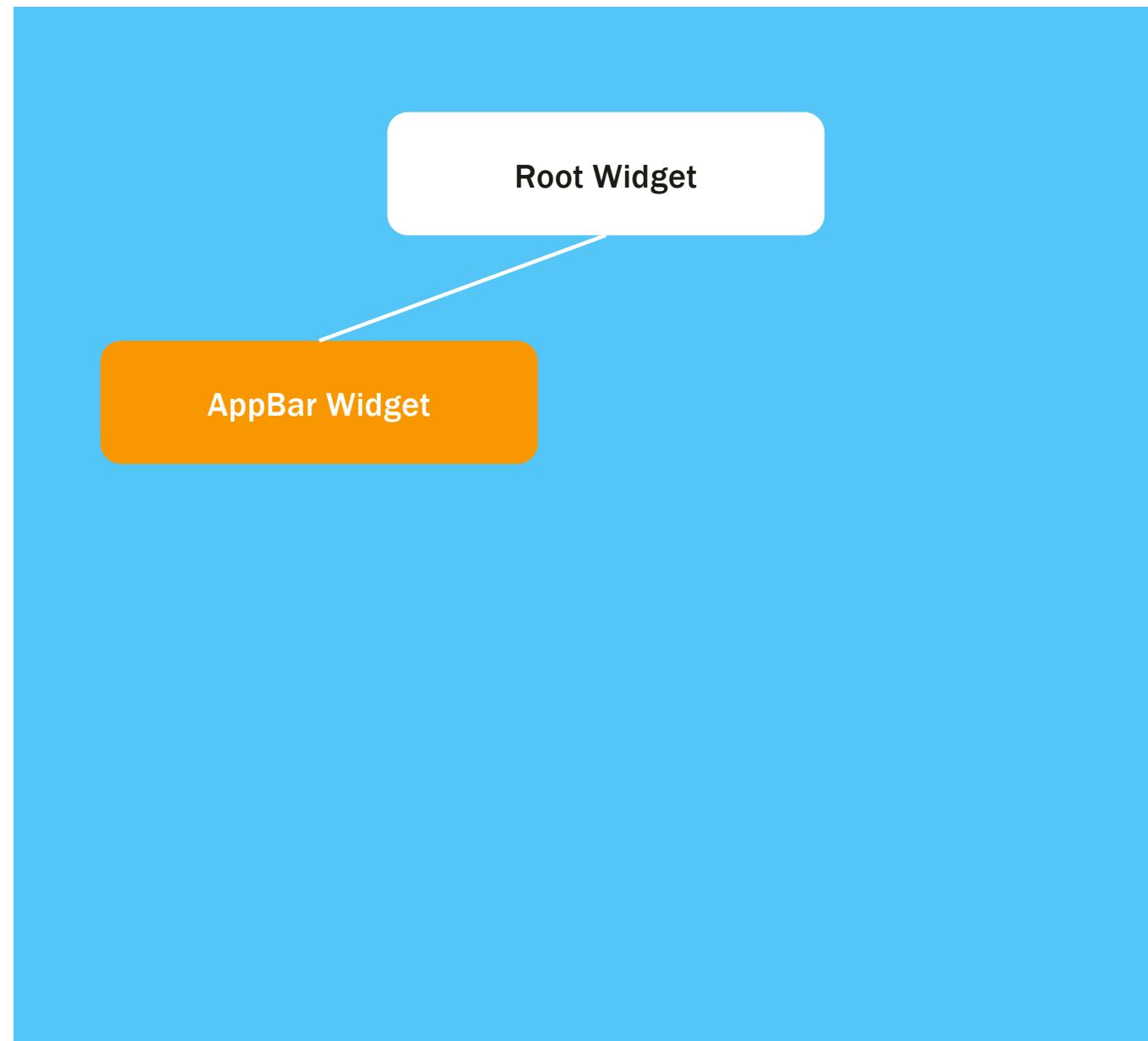
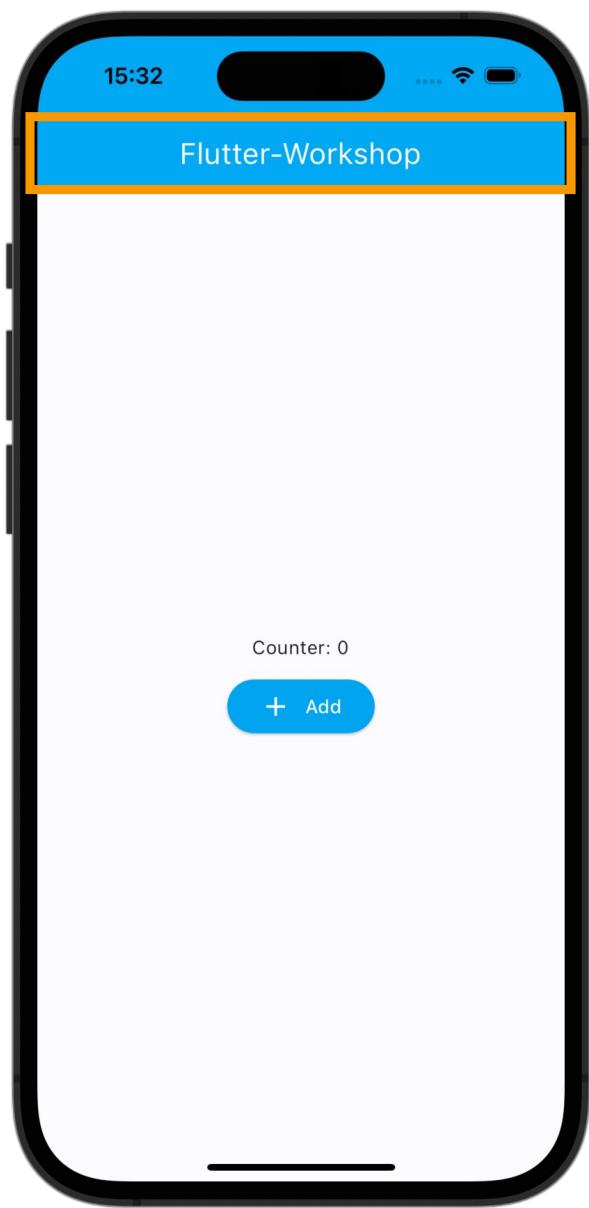
Widgets

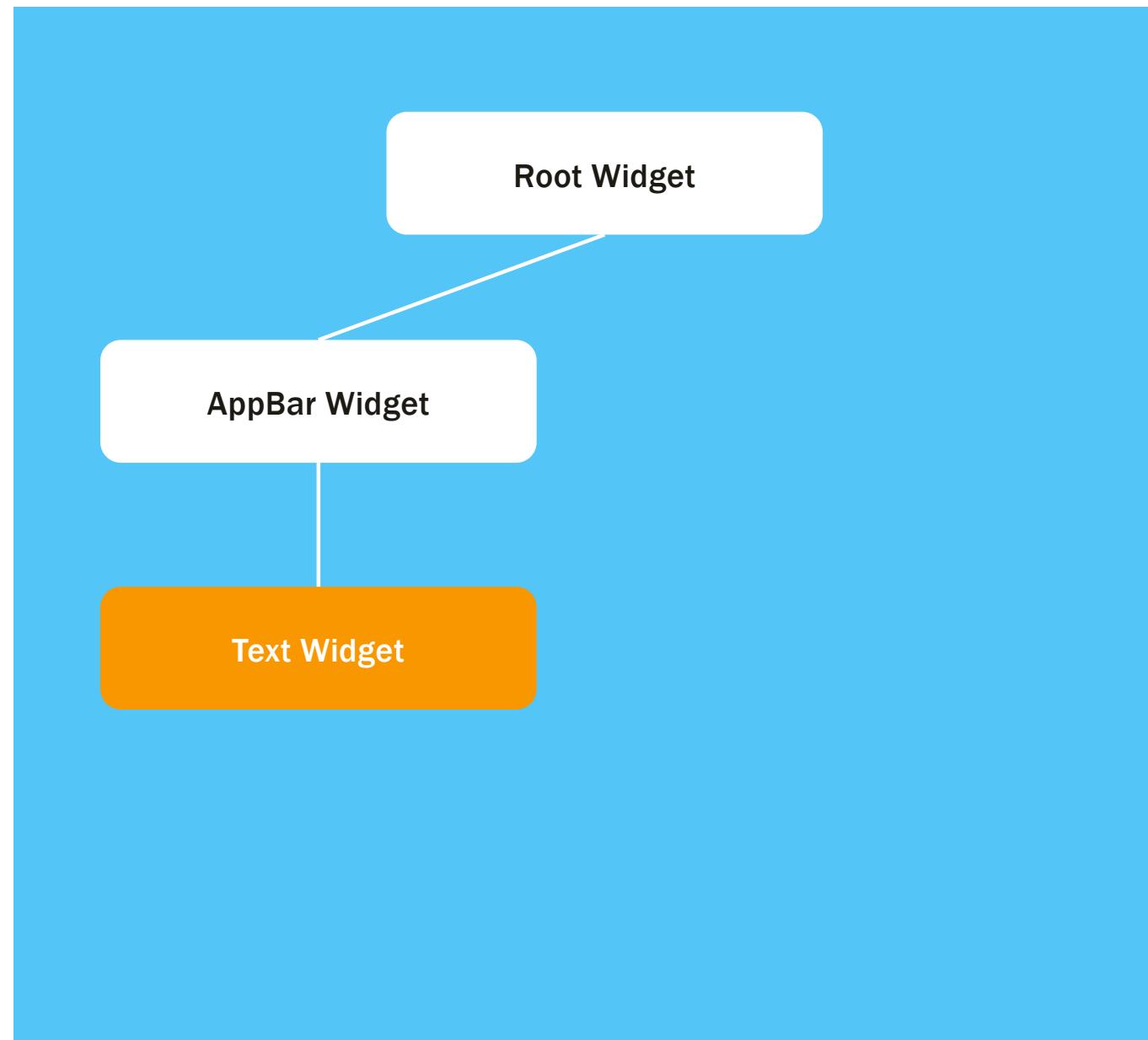
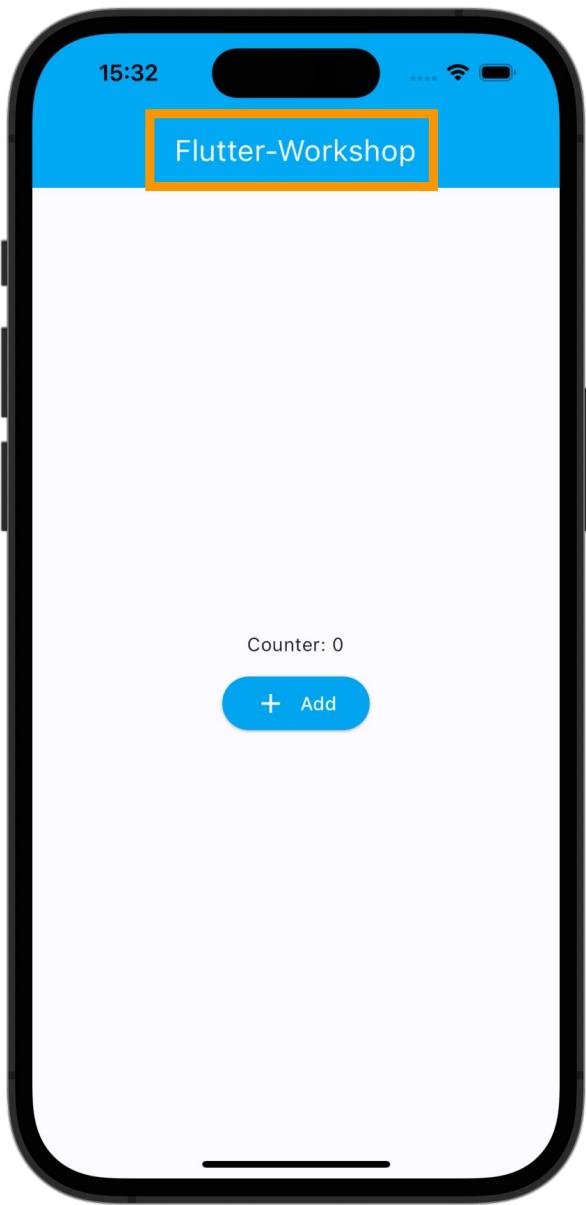


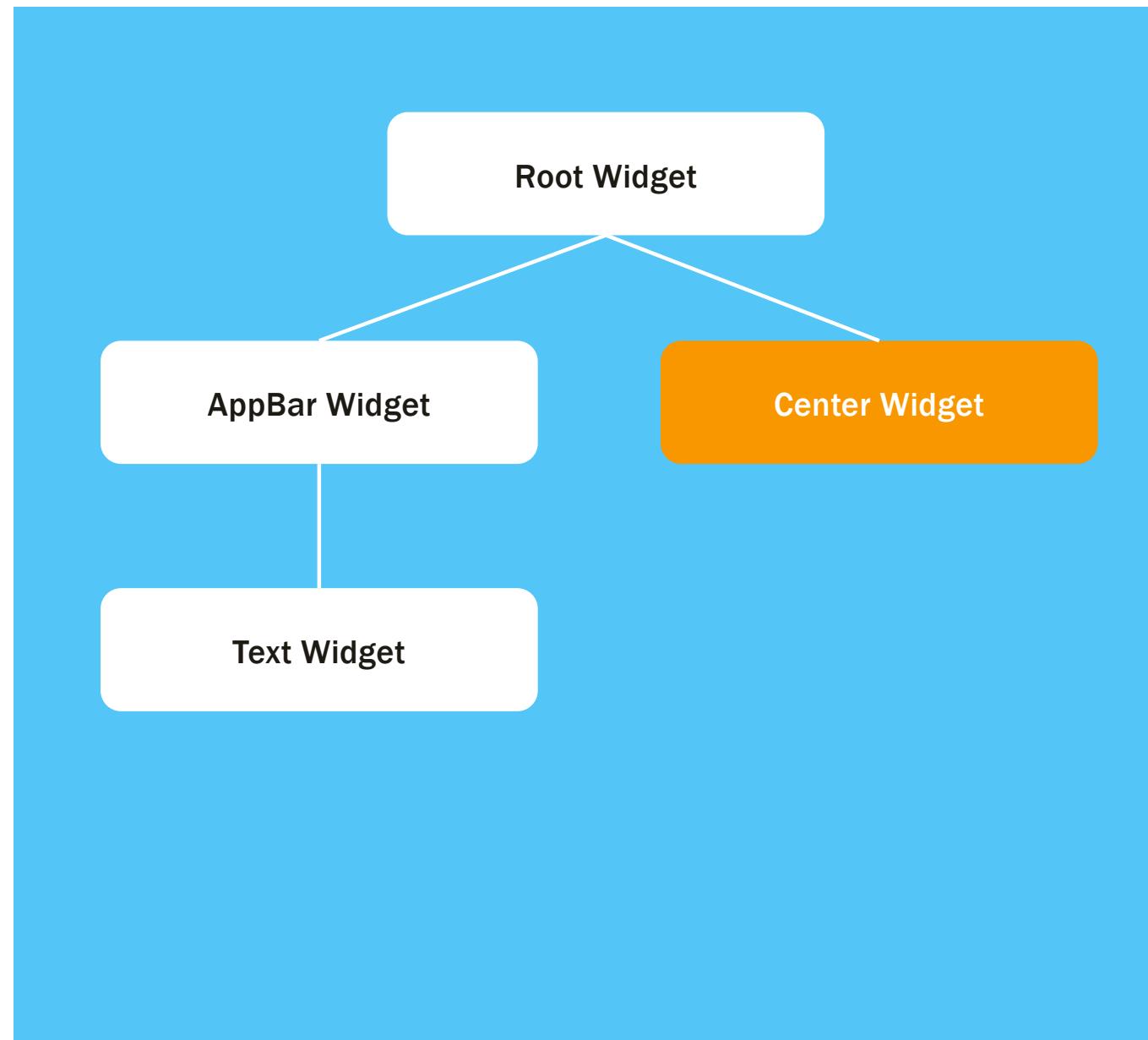
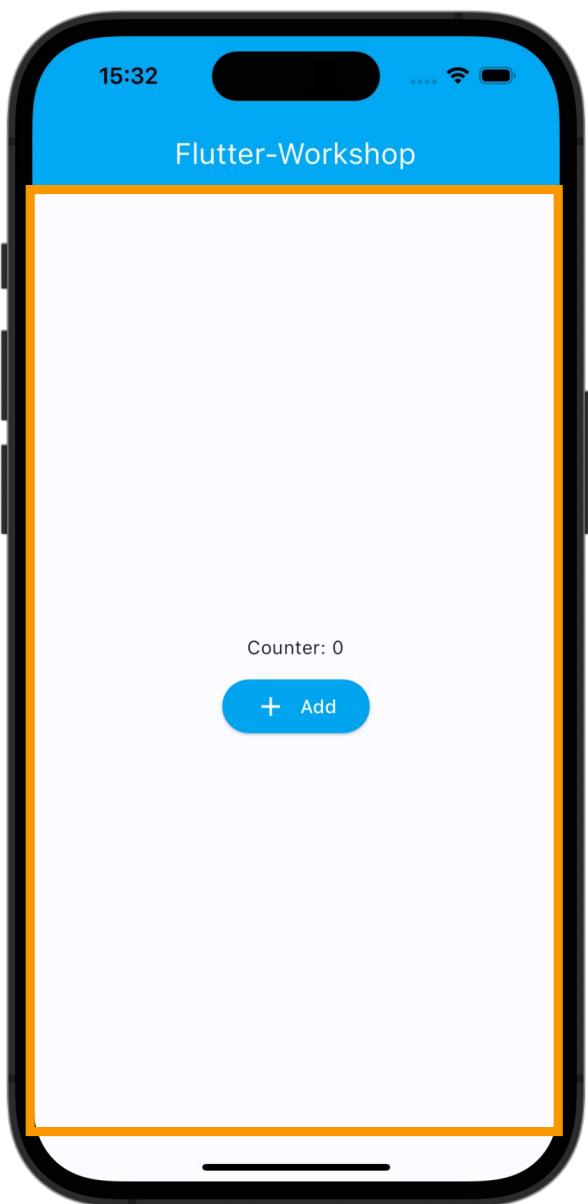


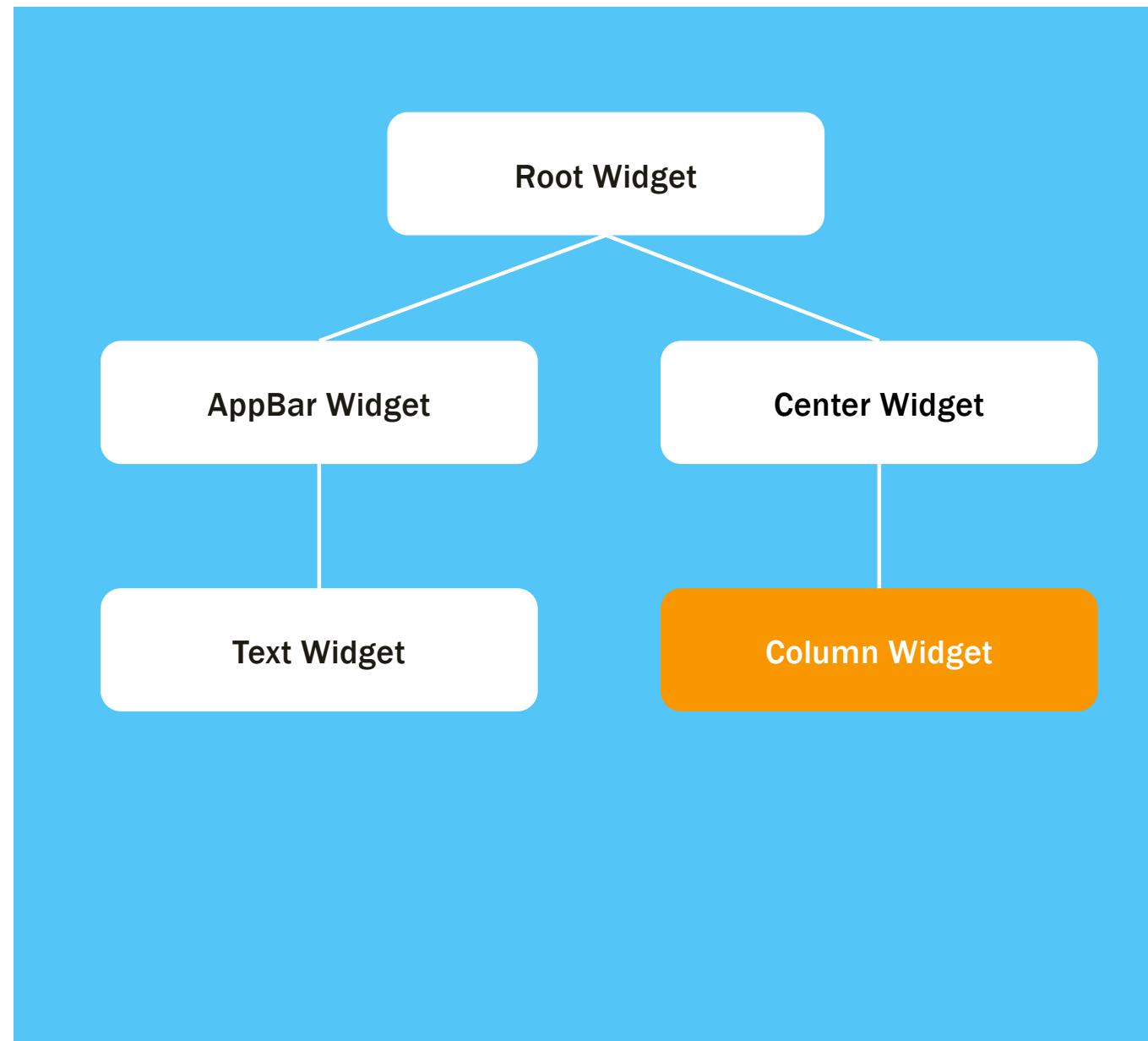
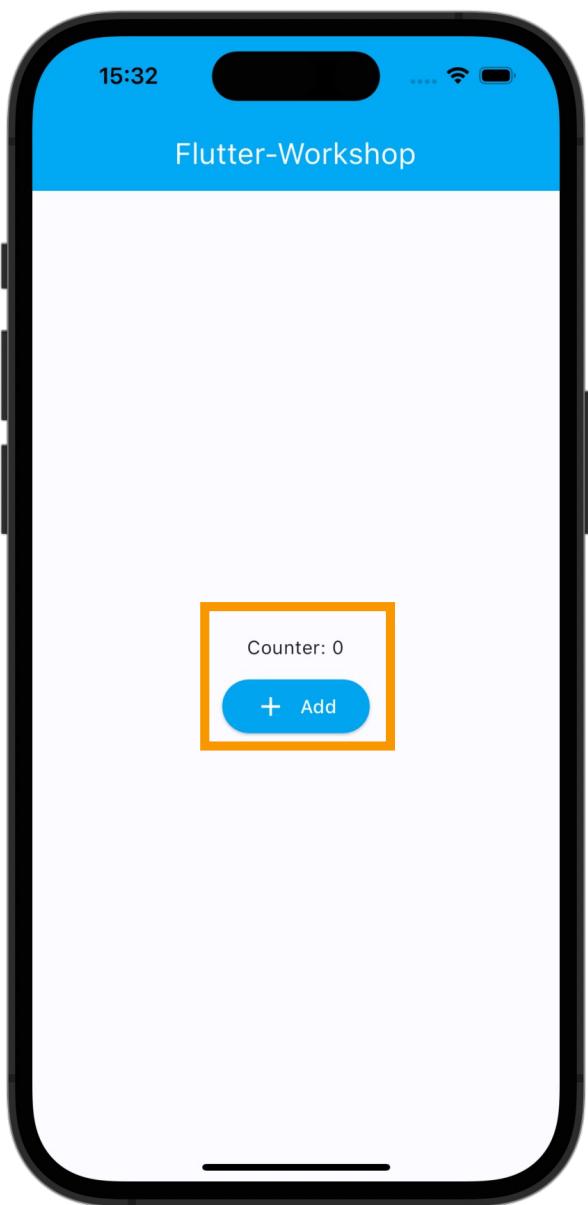


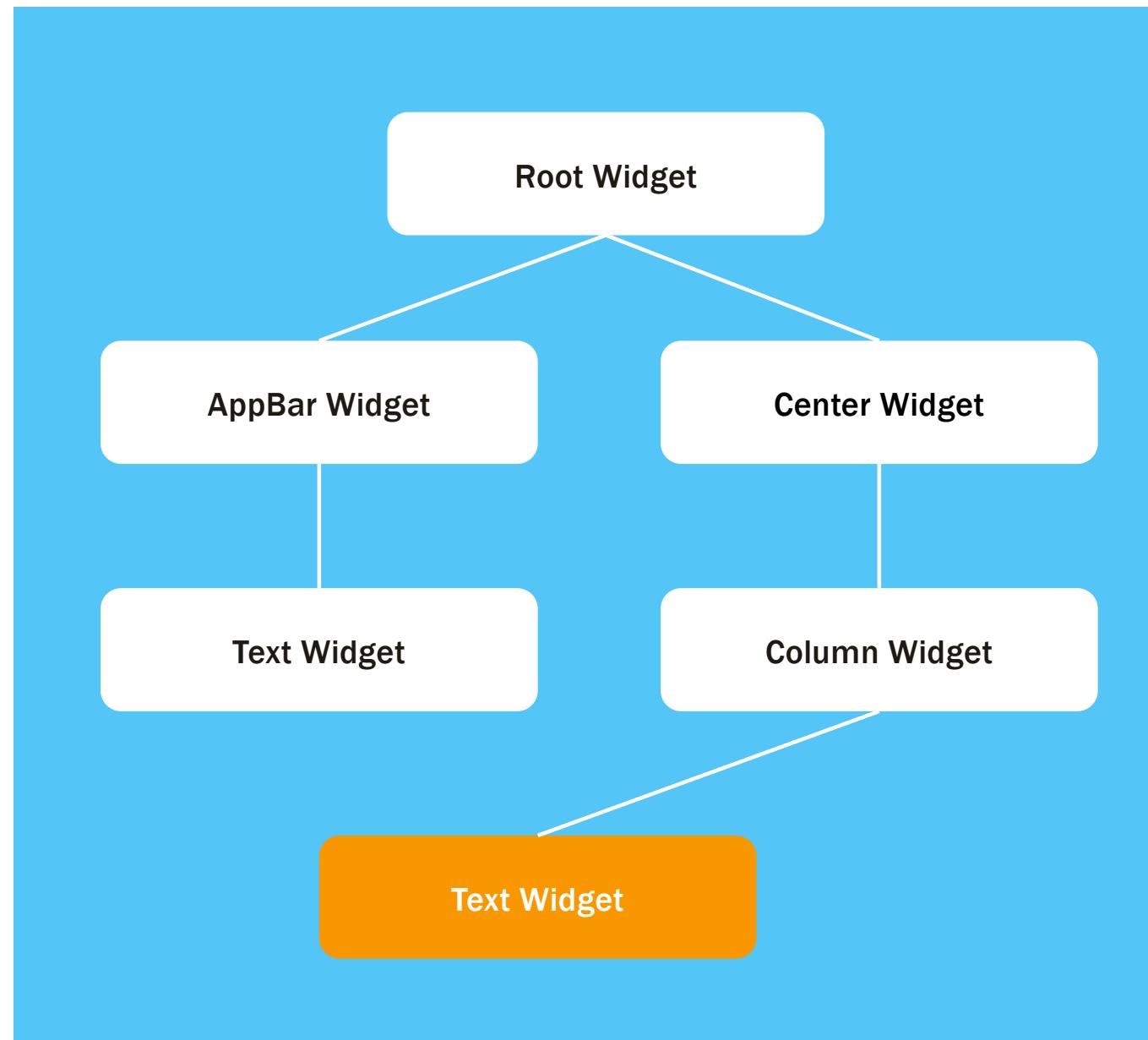
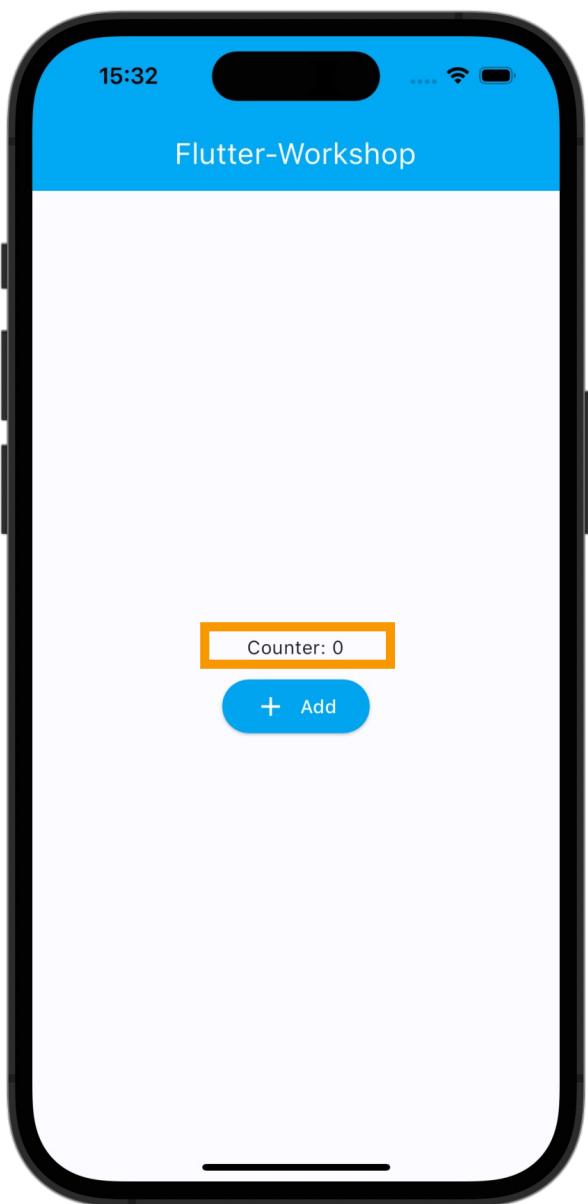


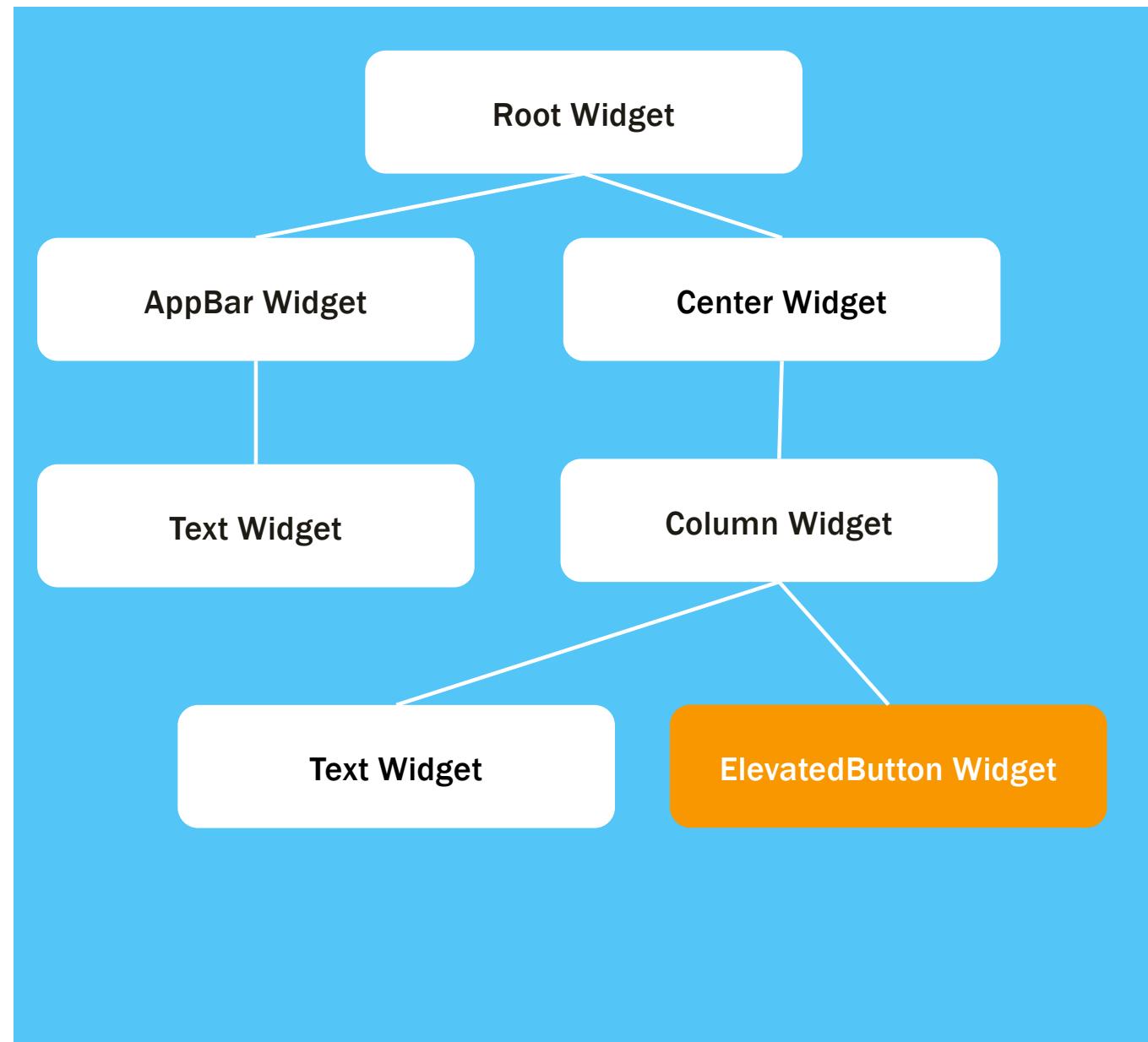
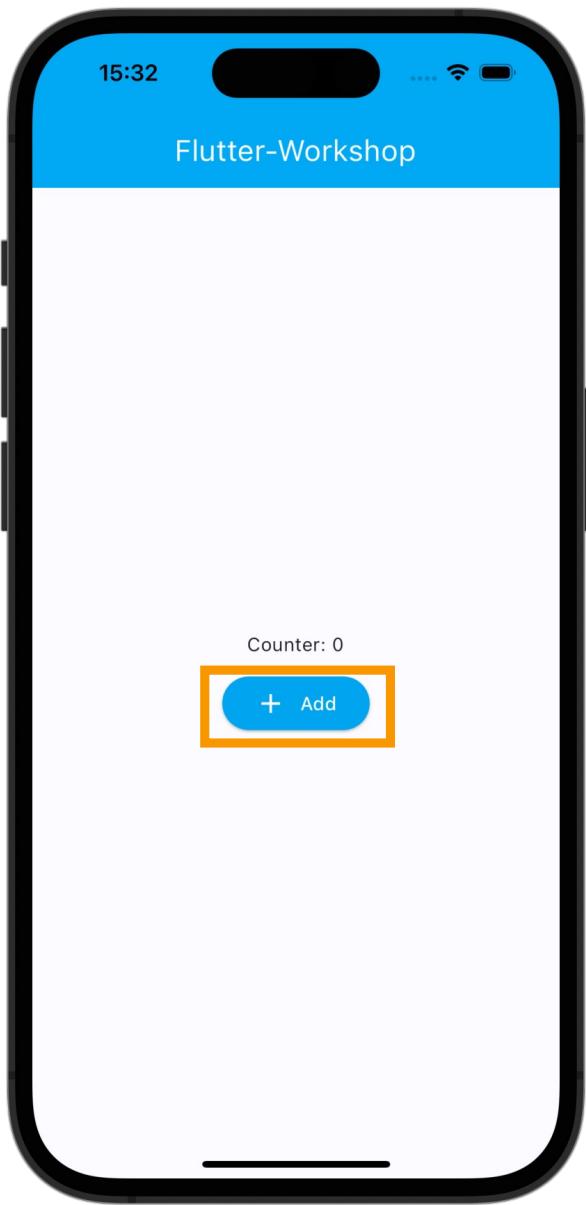


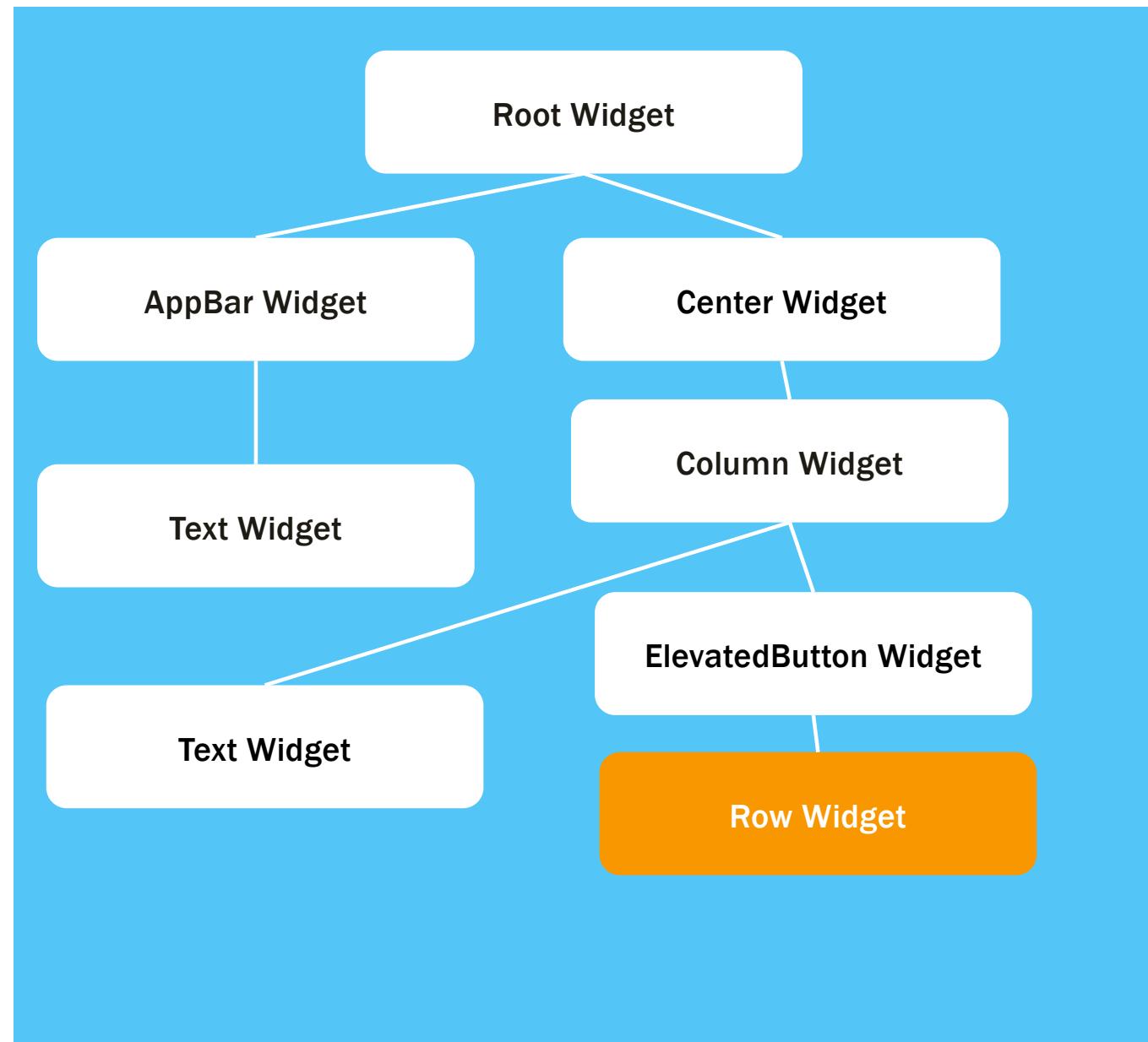
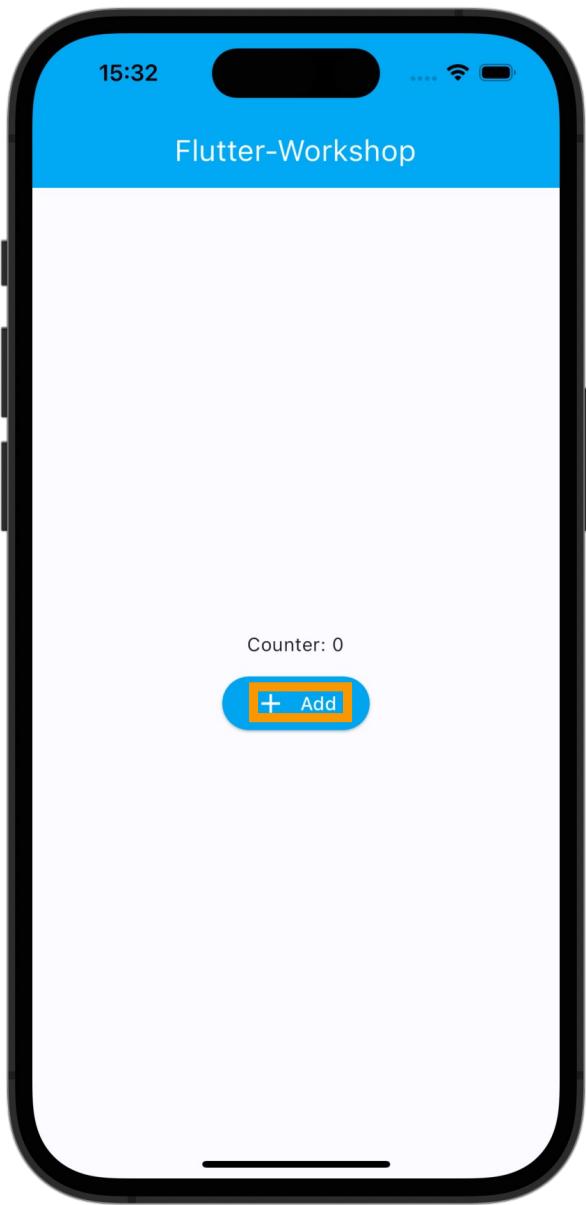


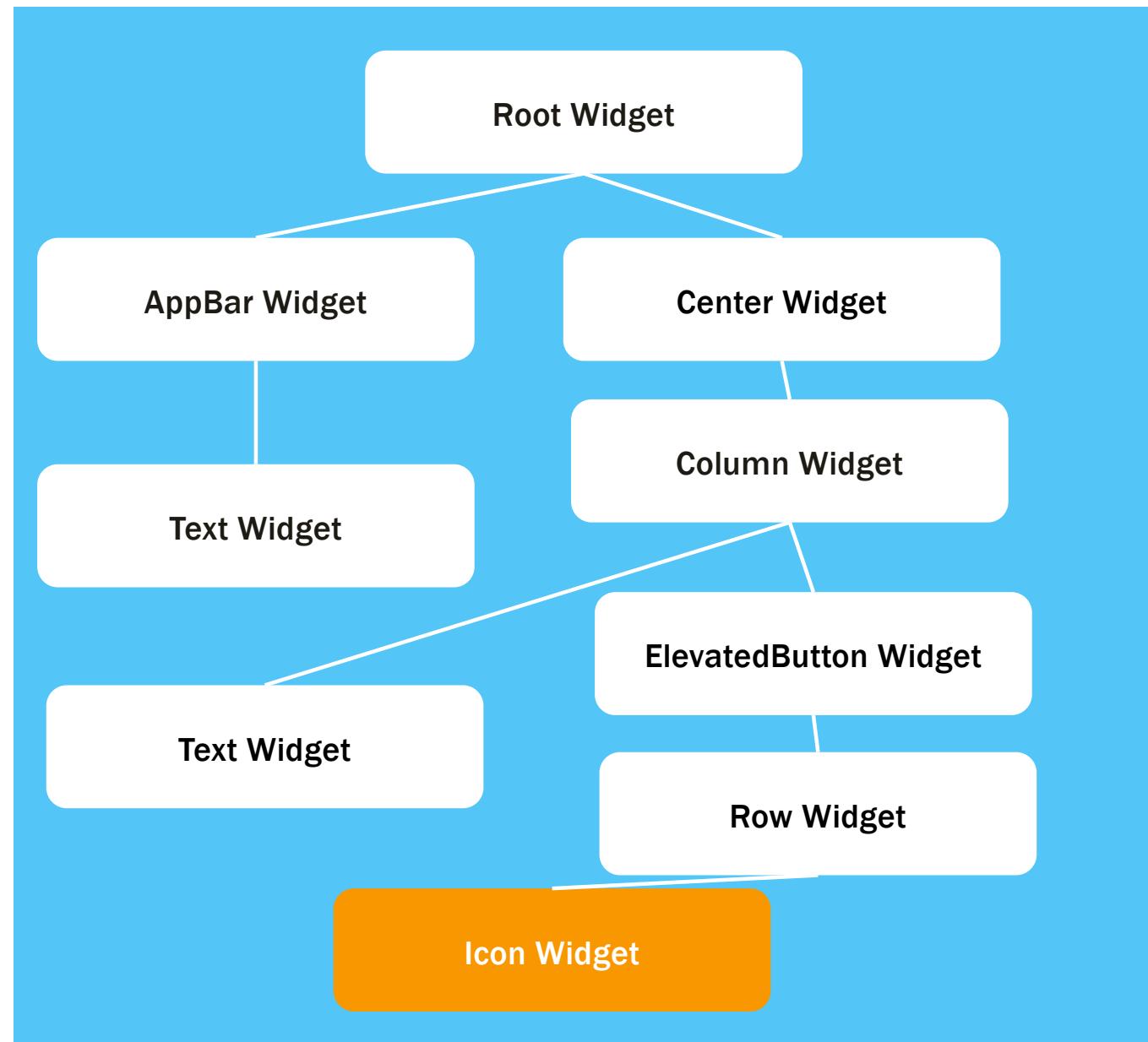
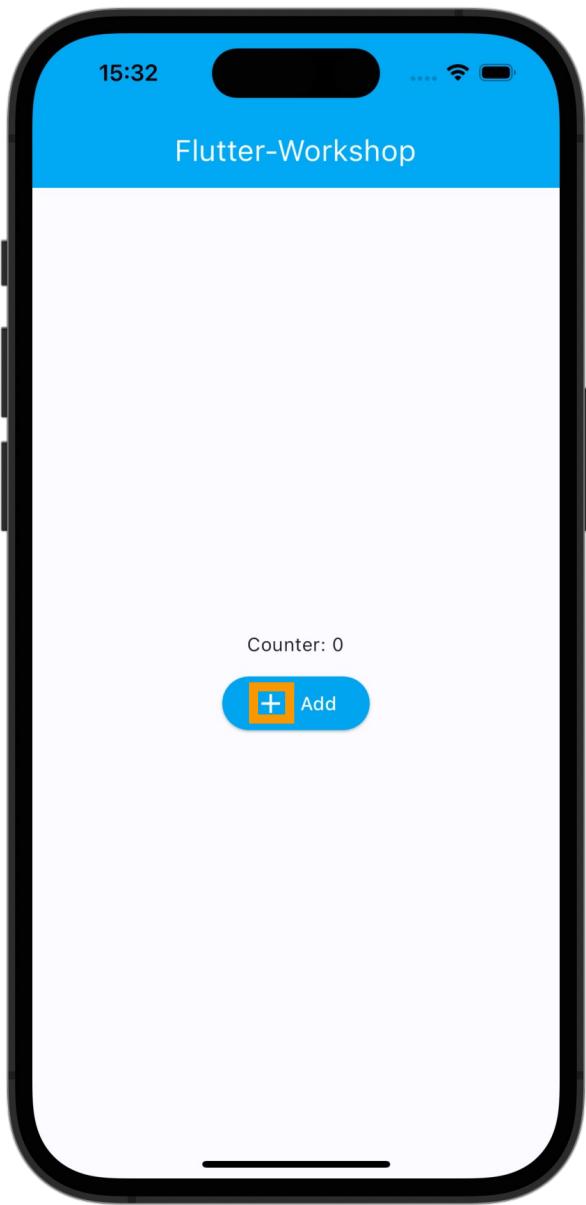


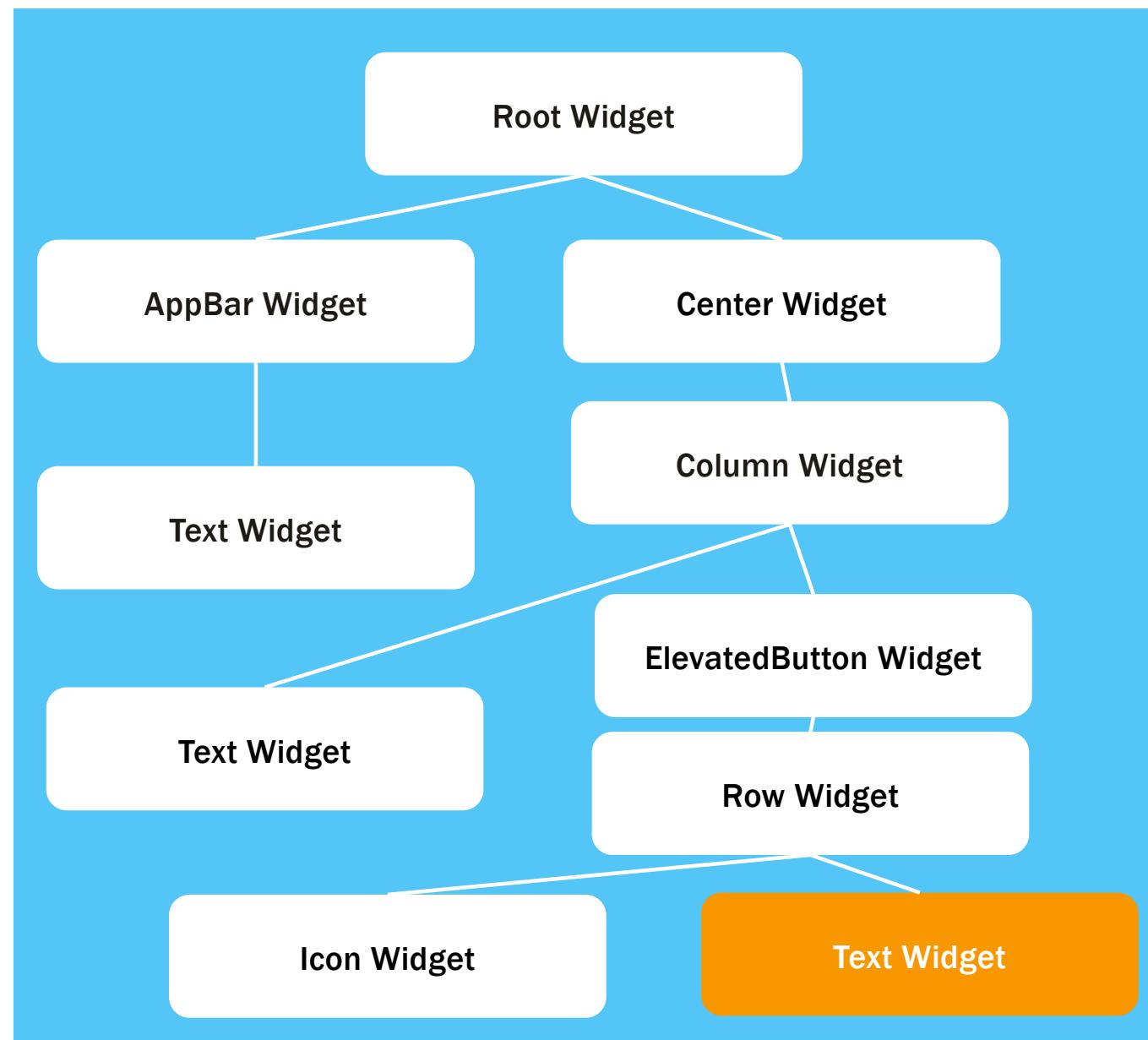
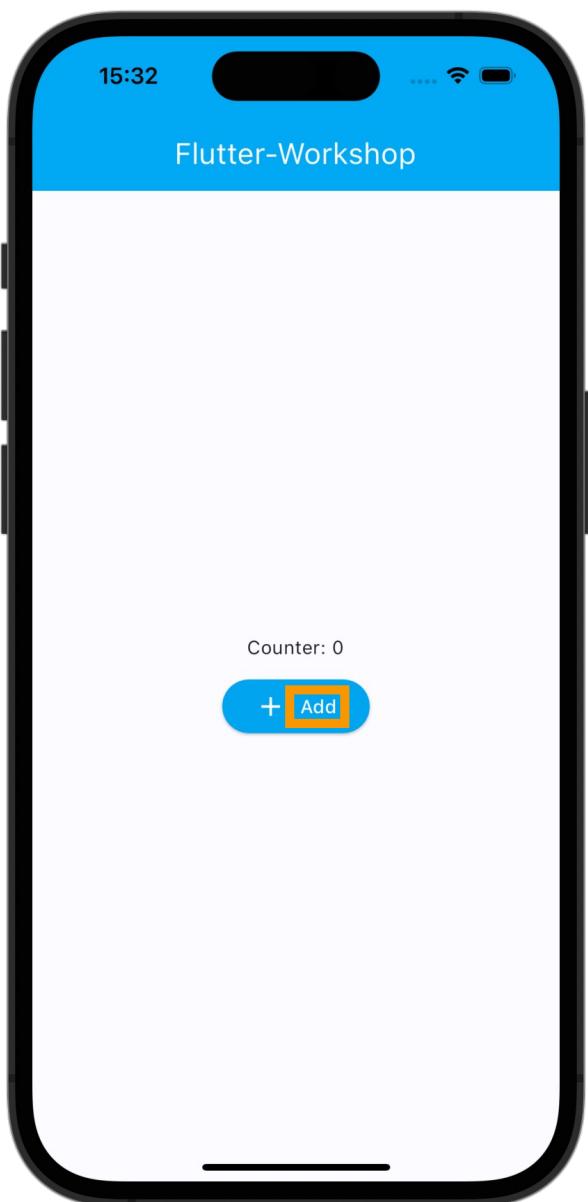


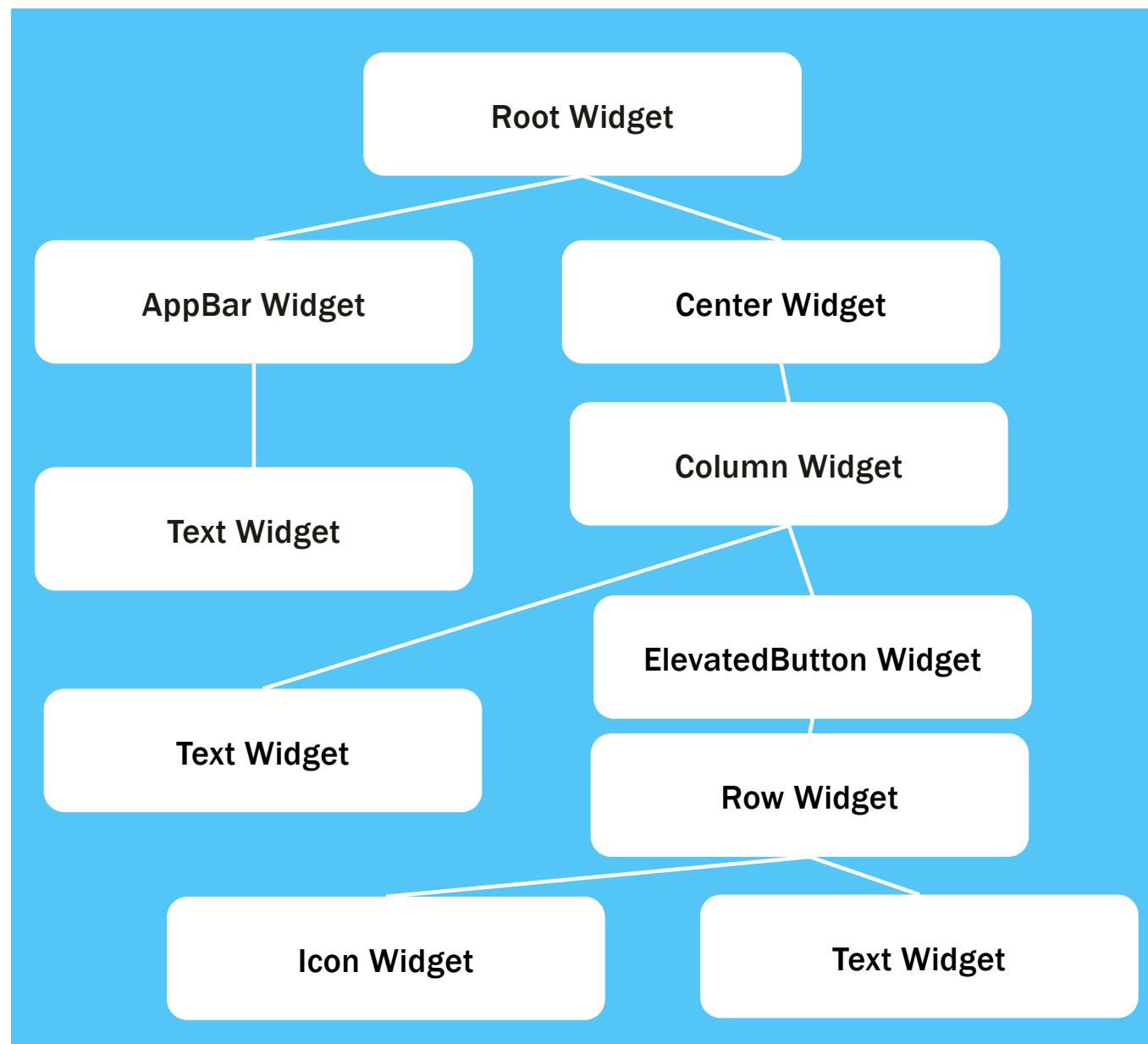
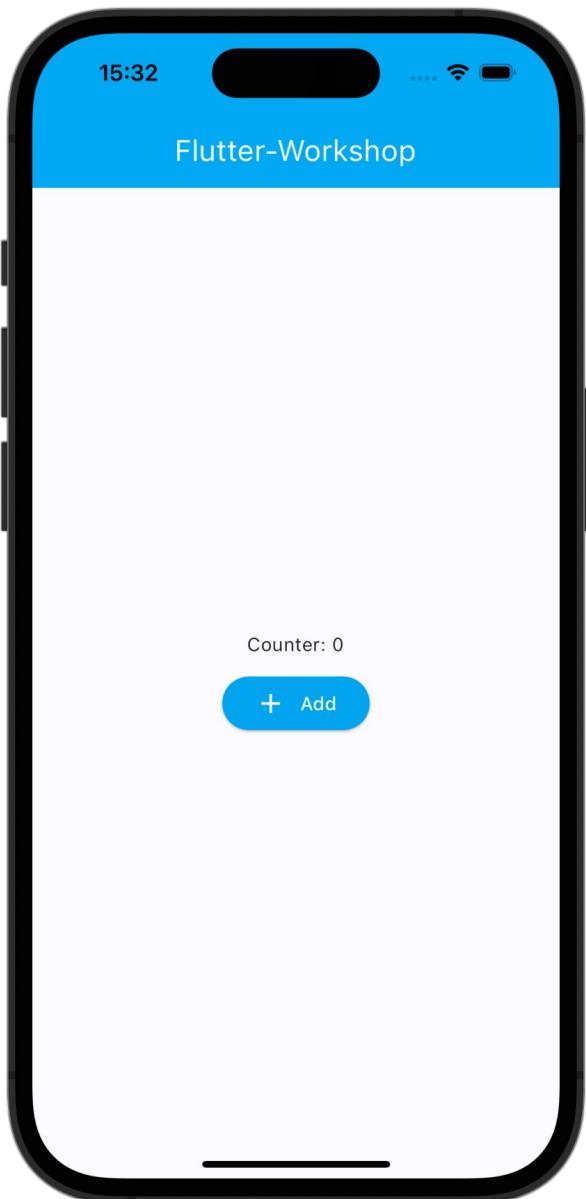












Widgets

Properties

Properties

Text Widget

style
textAlign
overflow
maxLines

...

ElevatedButton
Widget

style
onPressed

...

Column Widget

mainAxisAlignment
crossAxisAlignment

...

AppBar Widget

title
backgroundColor
leading

...

Widgets

Stateless vs. Stateful

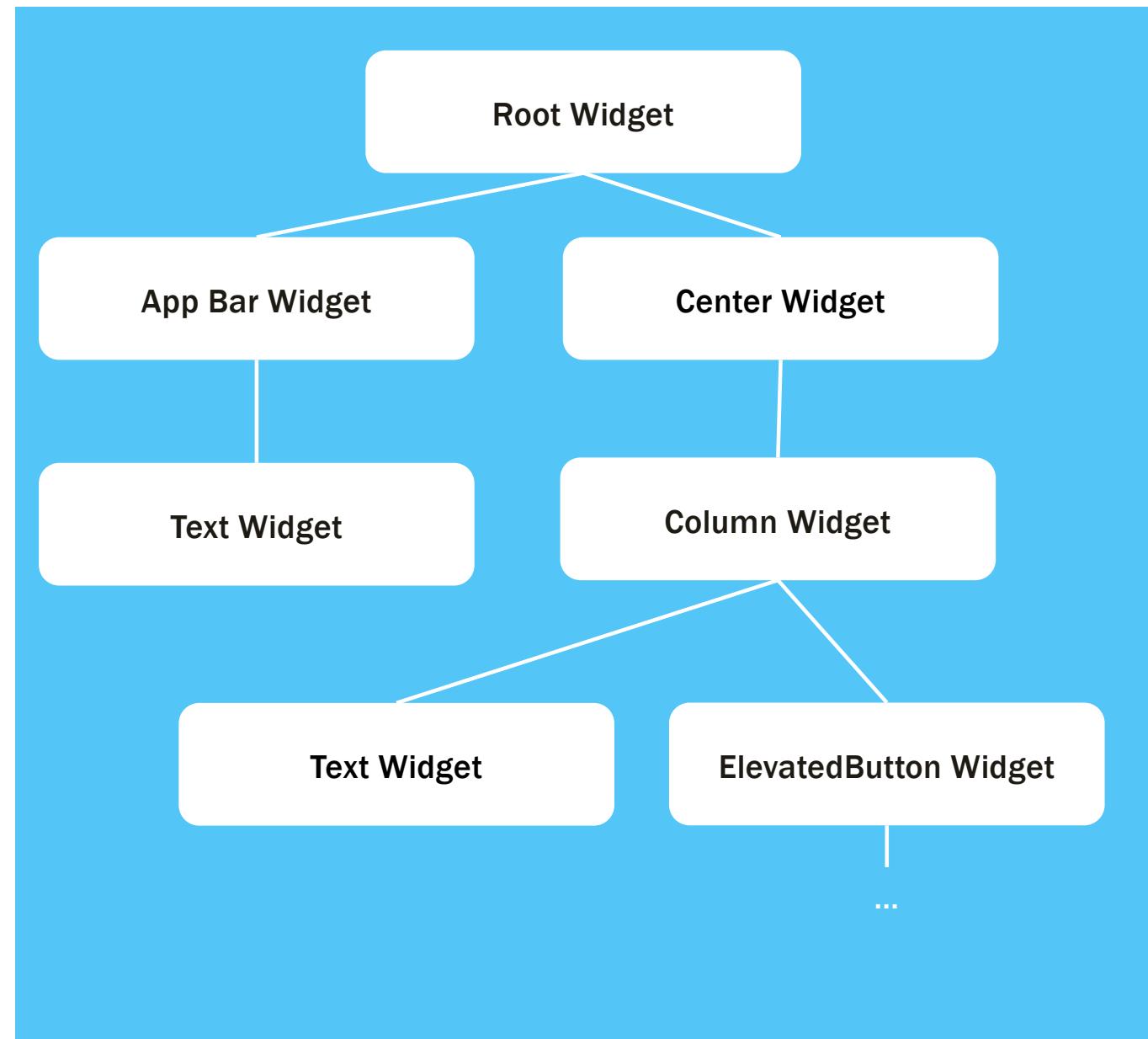
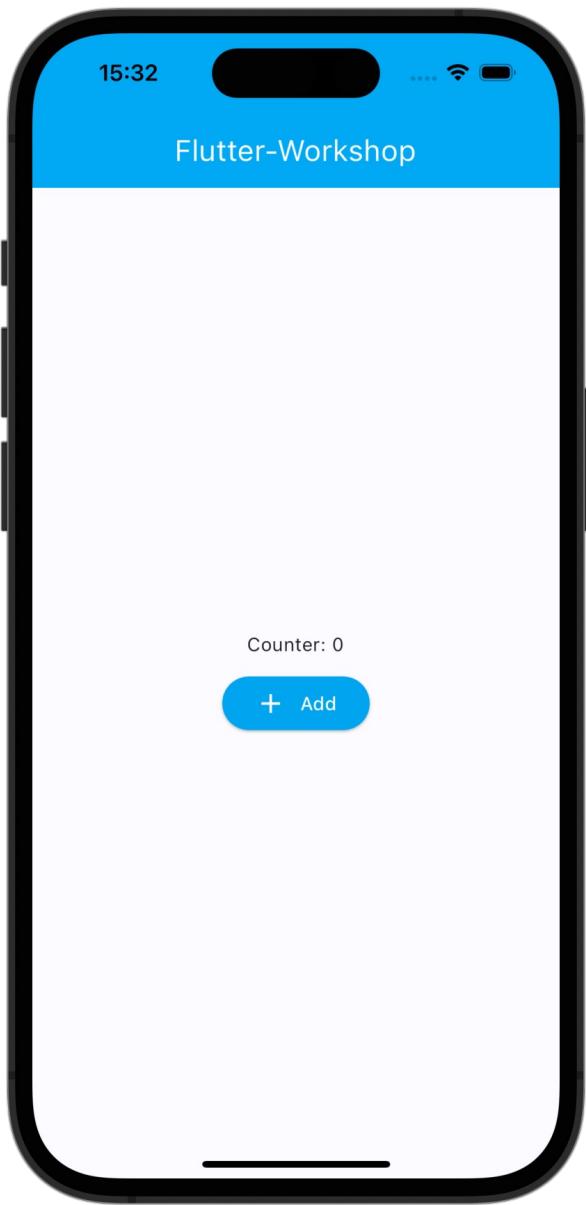
Stateless vs. Stateful

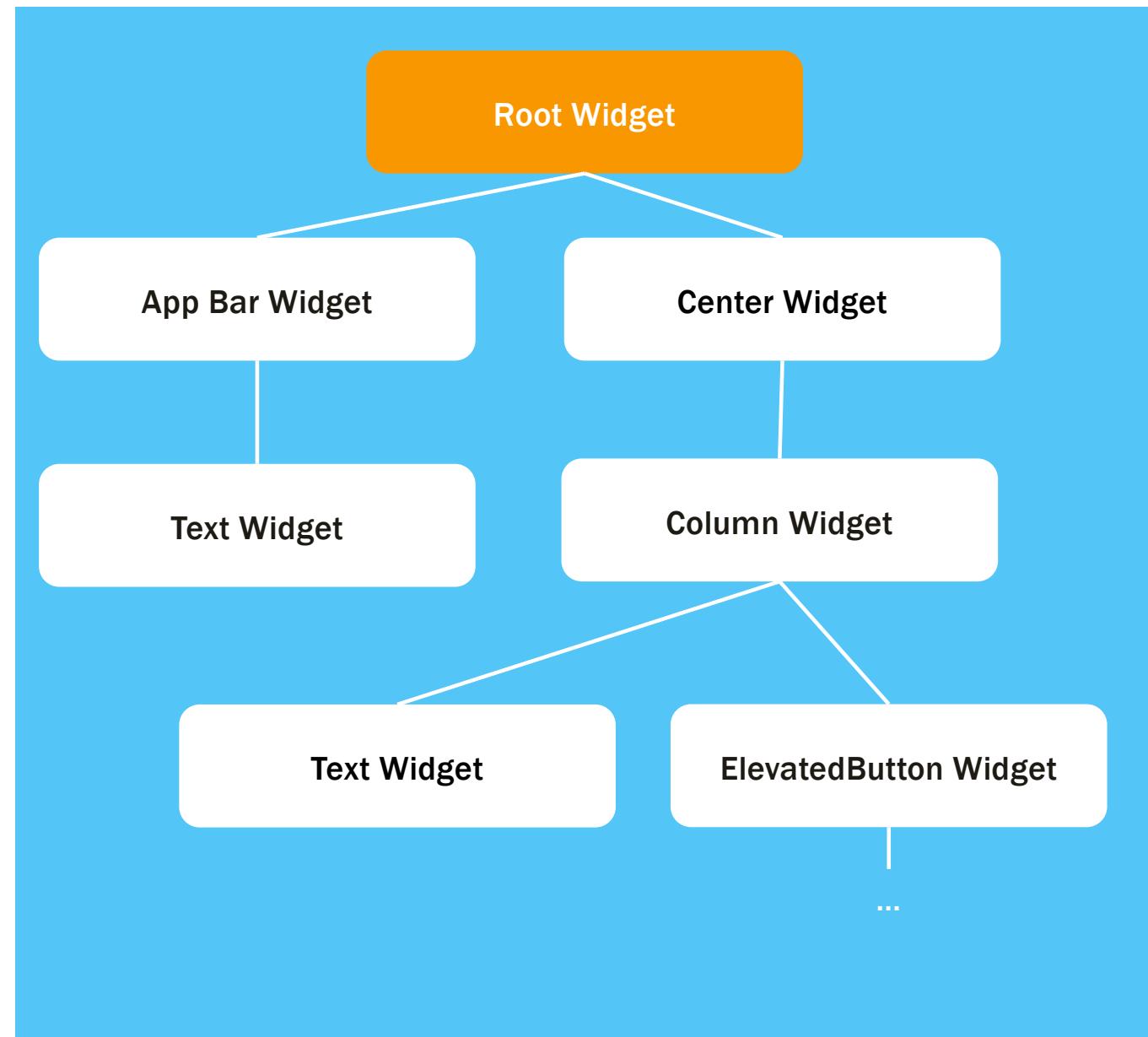
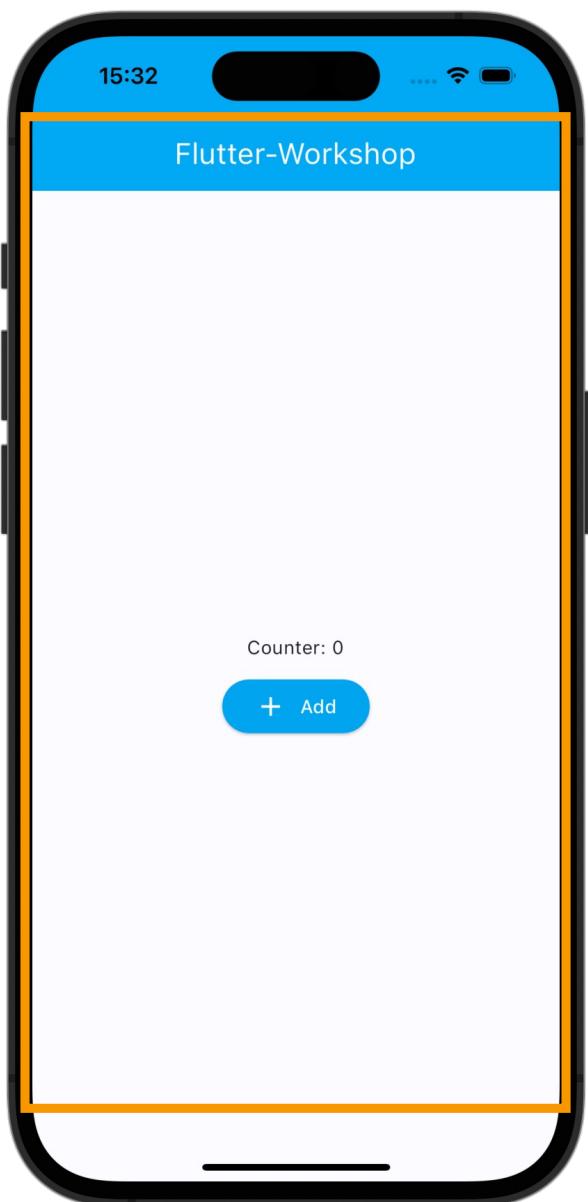
Stateless

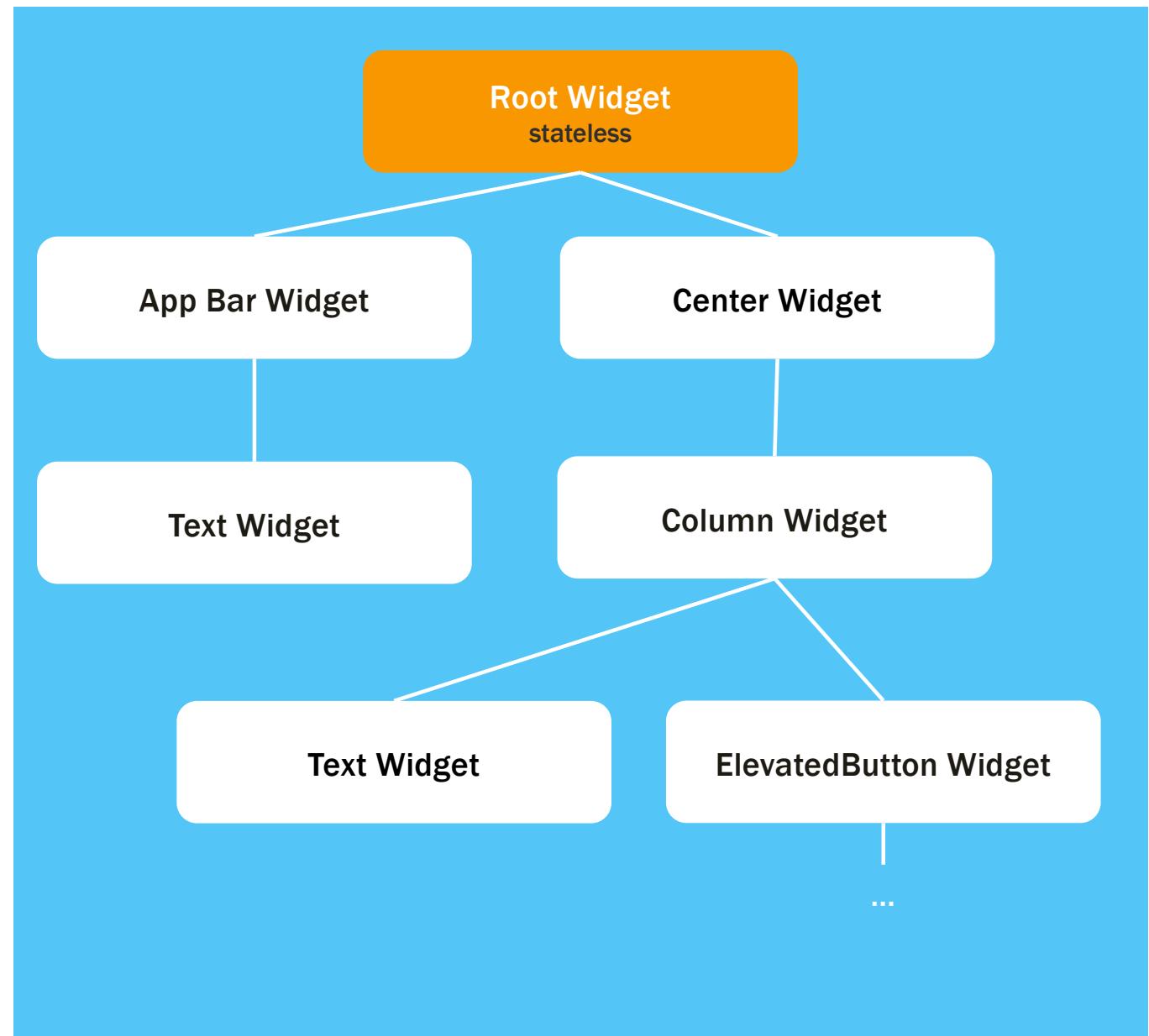
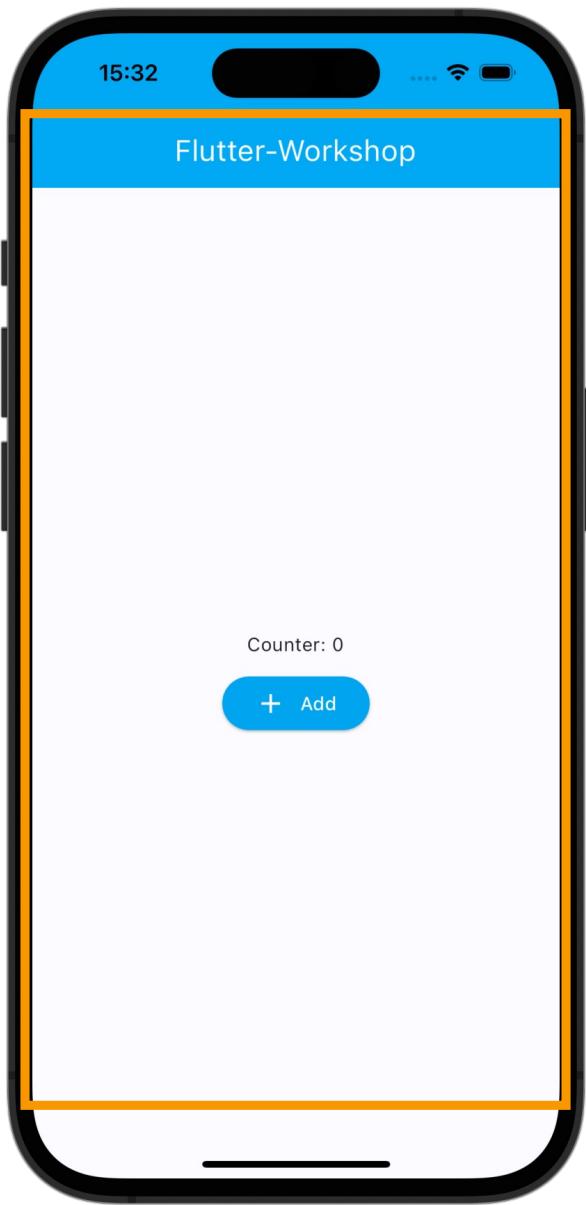
- “A widget that does not require mutable state.“
- Only needs to rerender when state in an ancestor widget changes

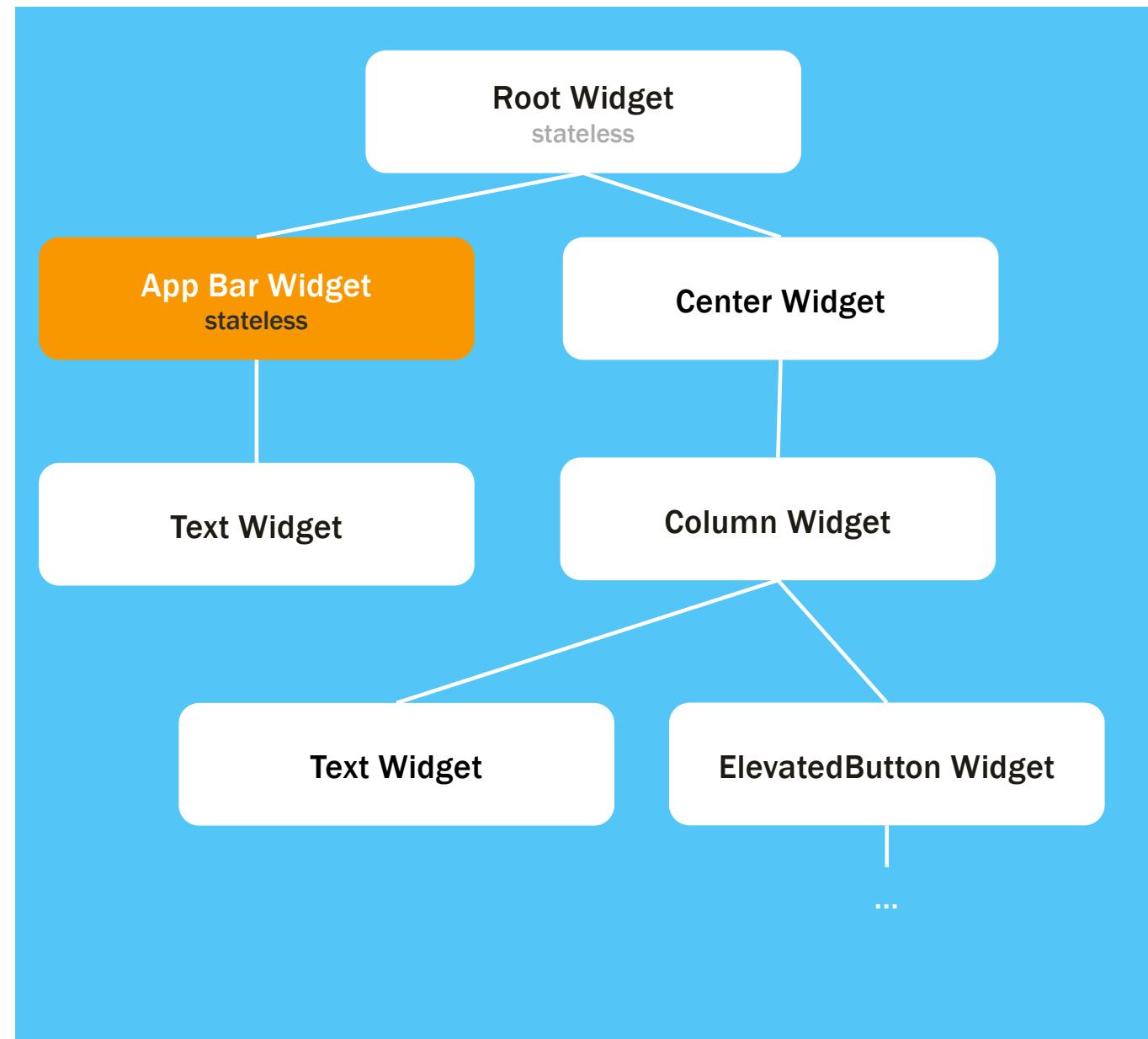
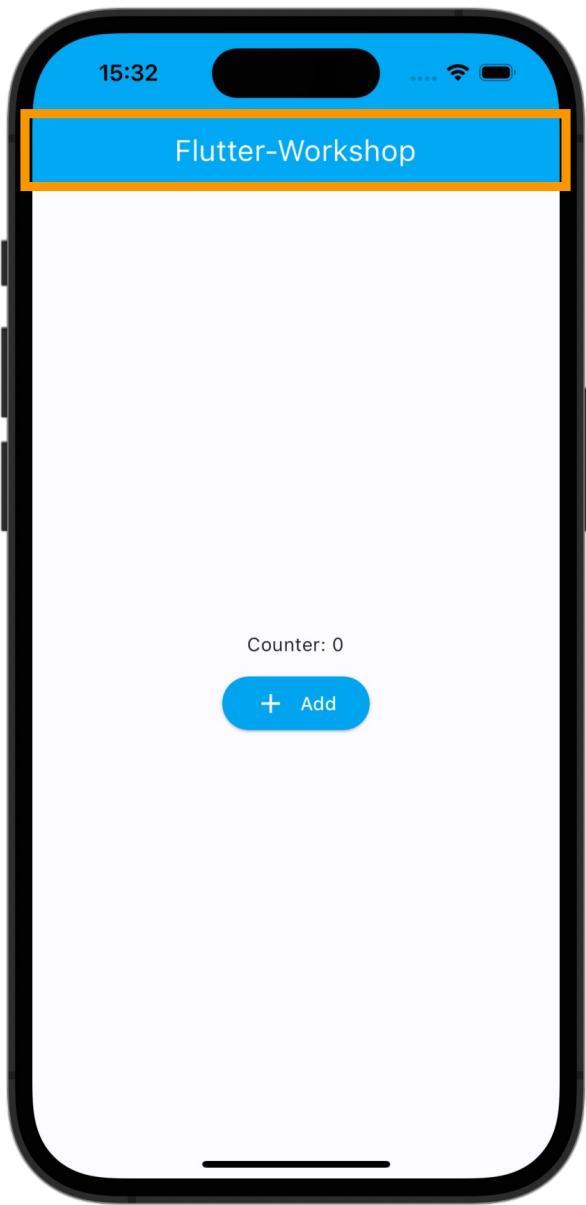
Stateful

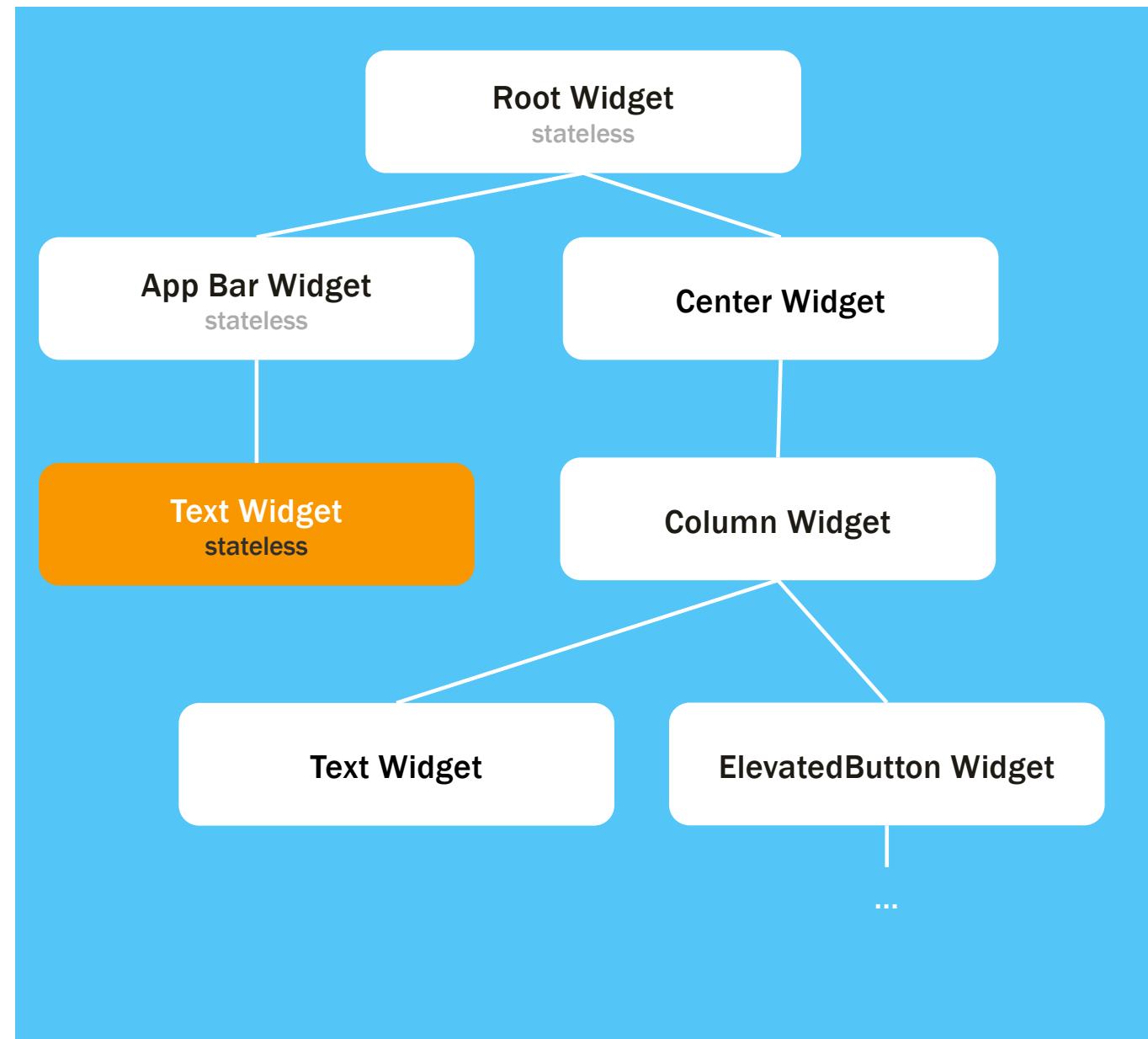
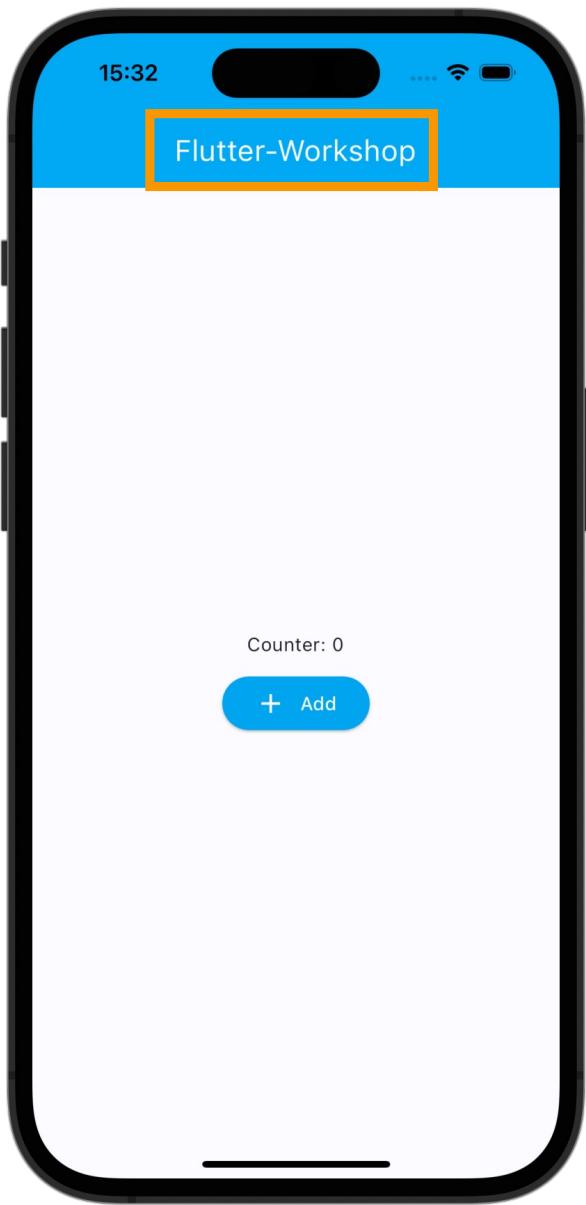
- “A widget that has mutable state.“
- Rerenders itself and all children every time its state changes

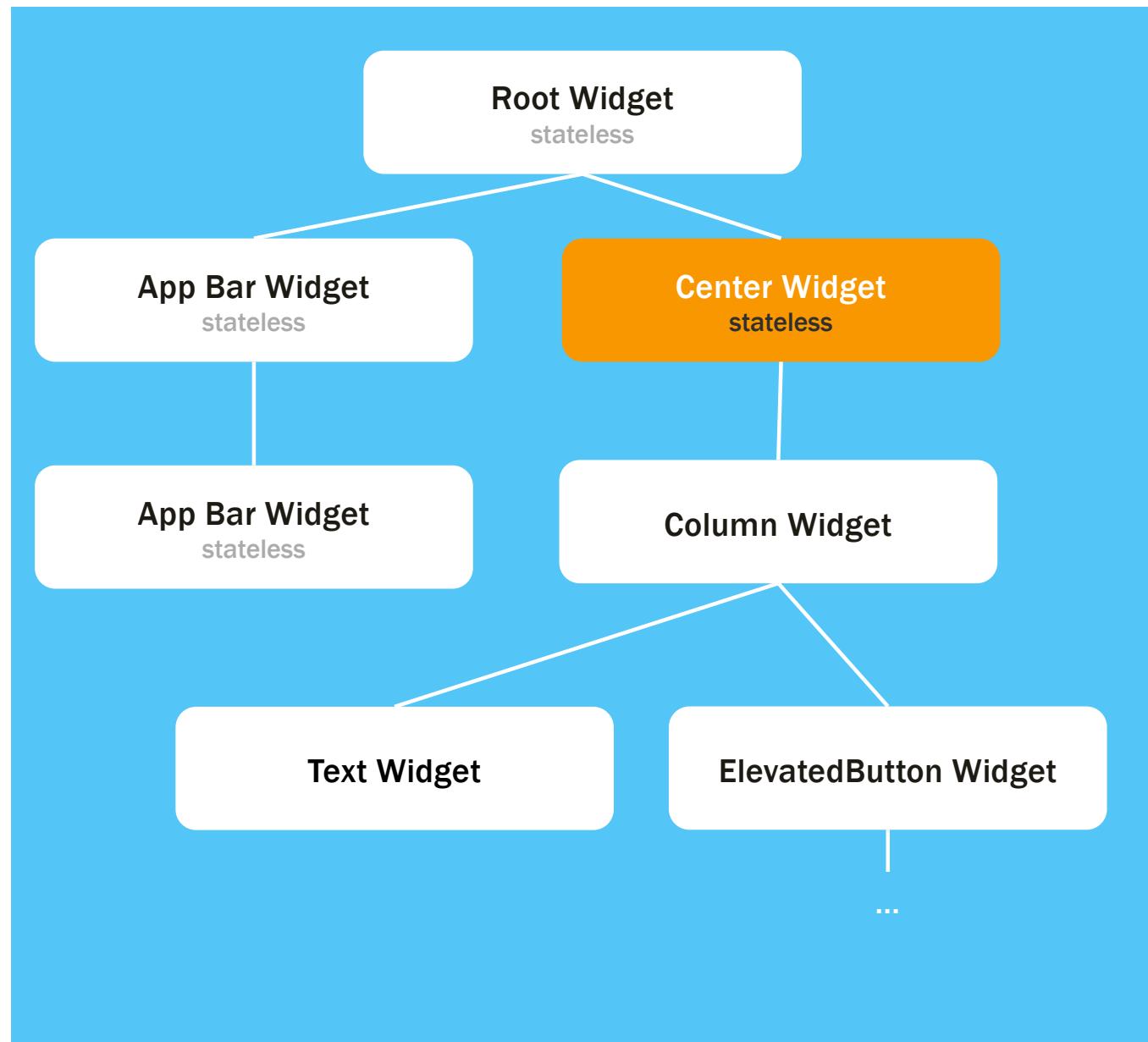
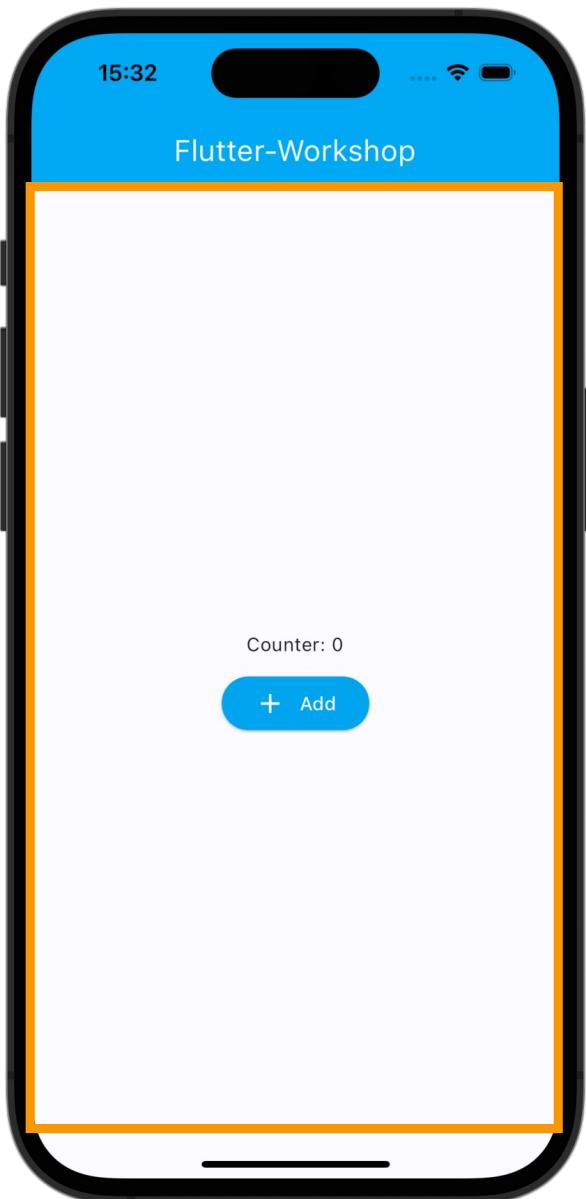


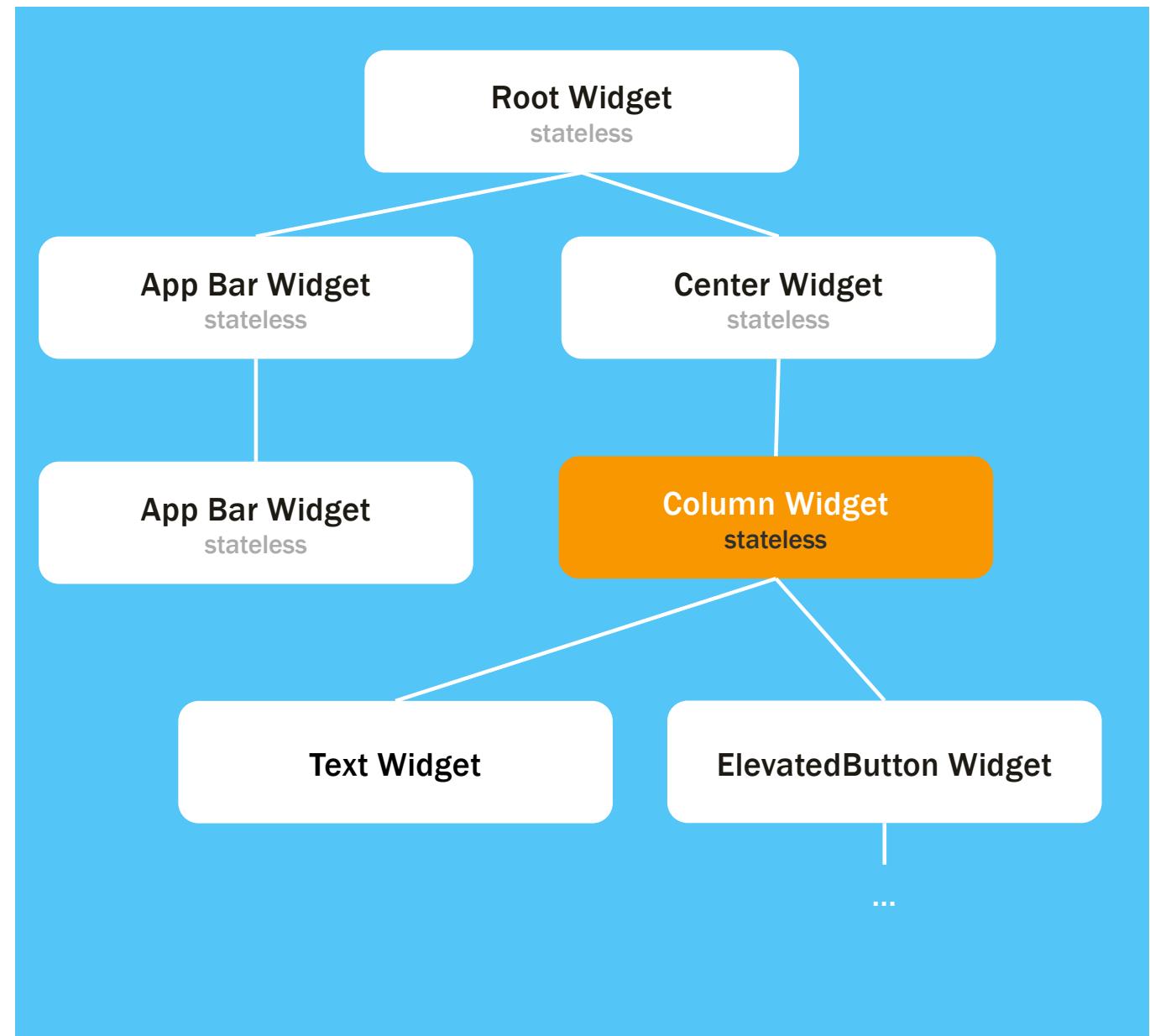
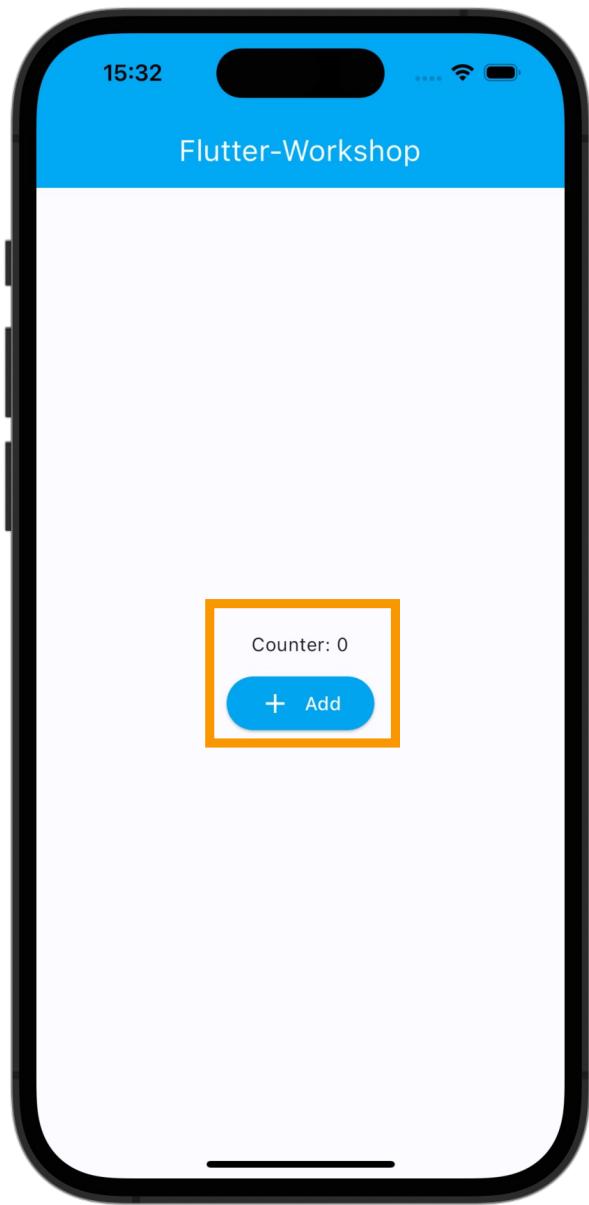


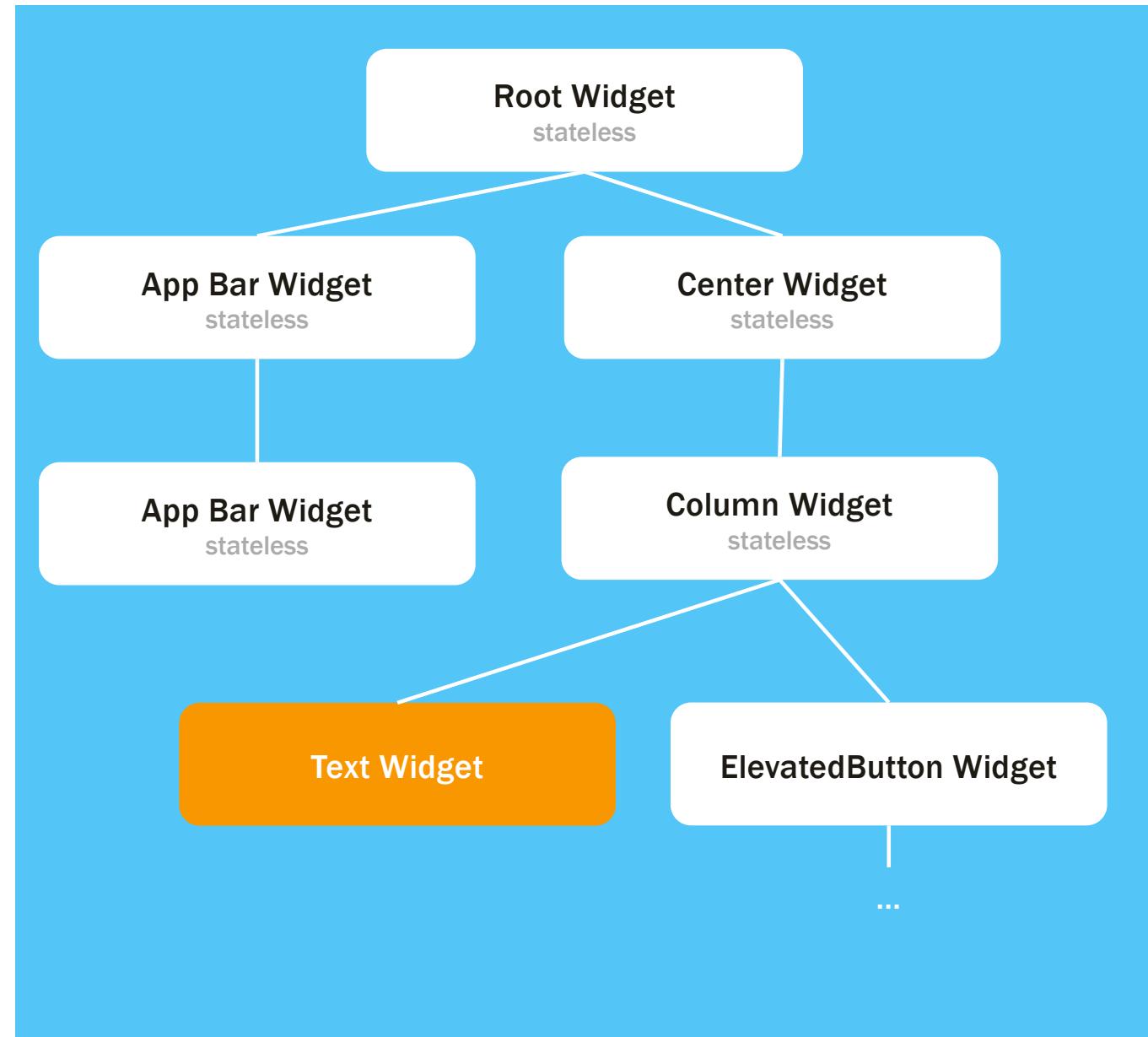
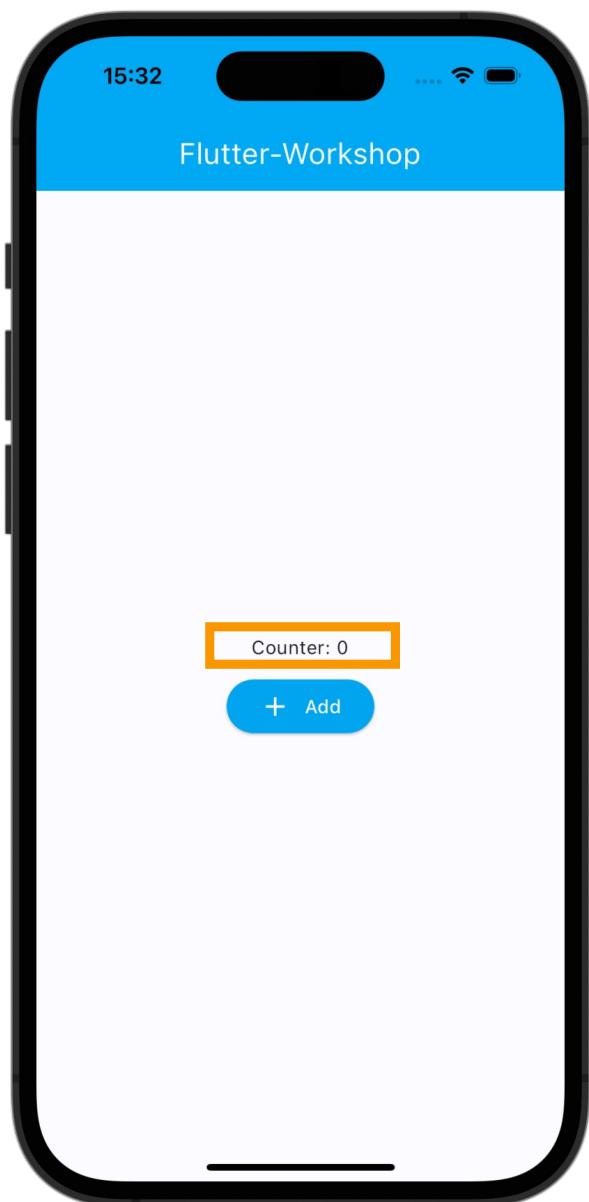


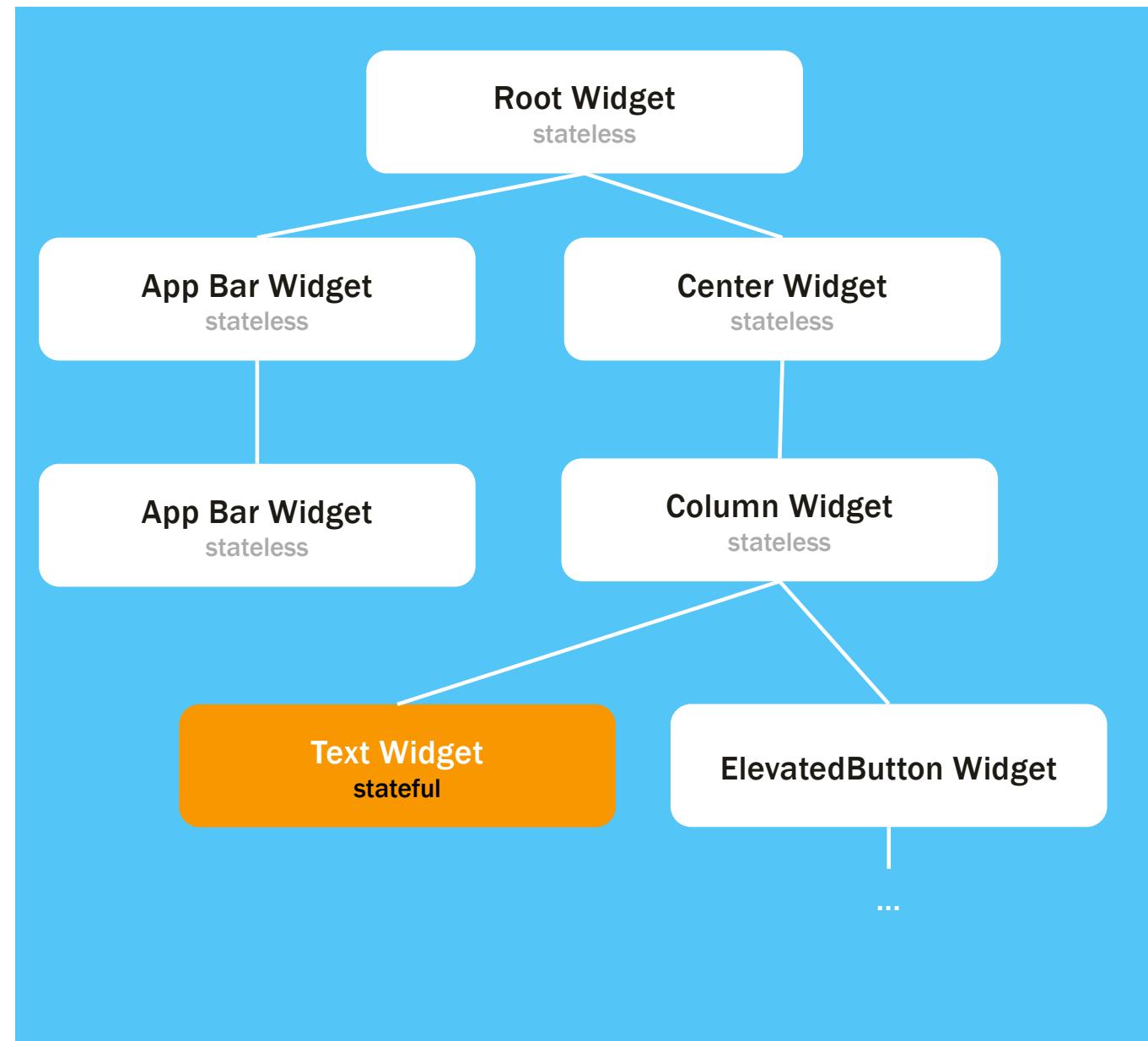
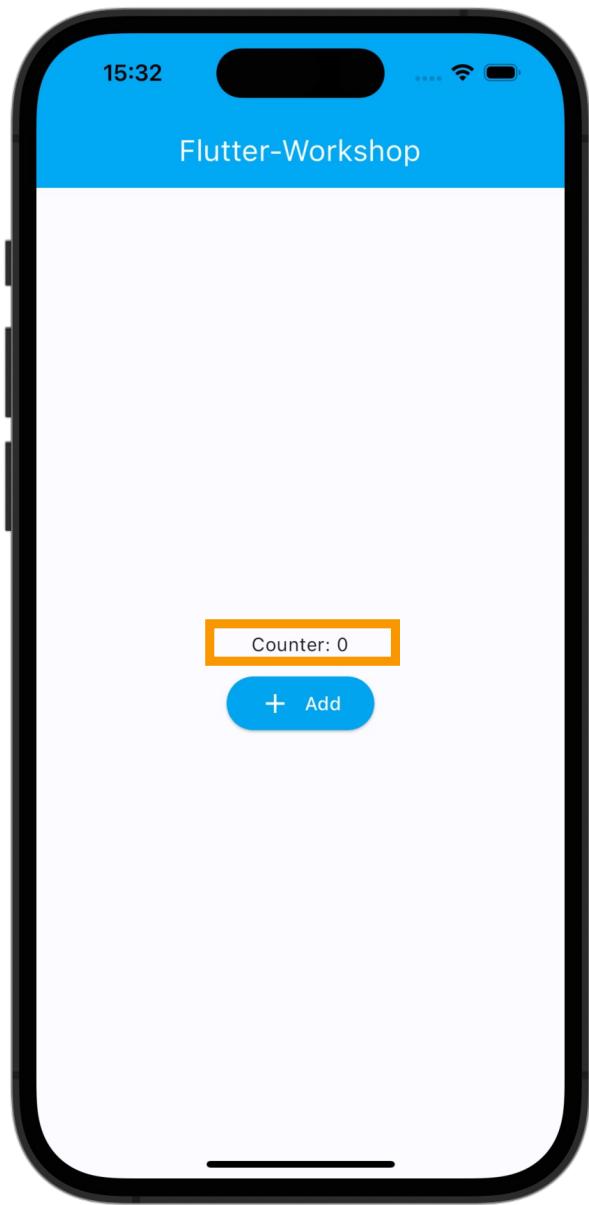


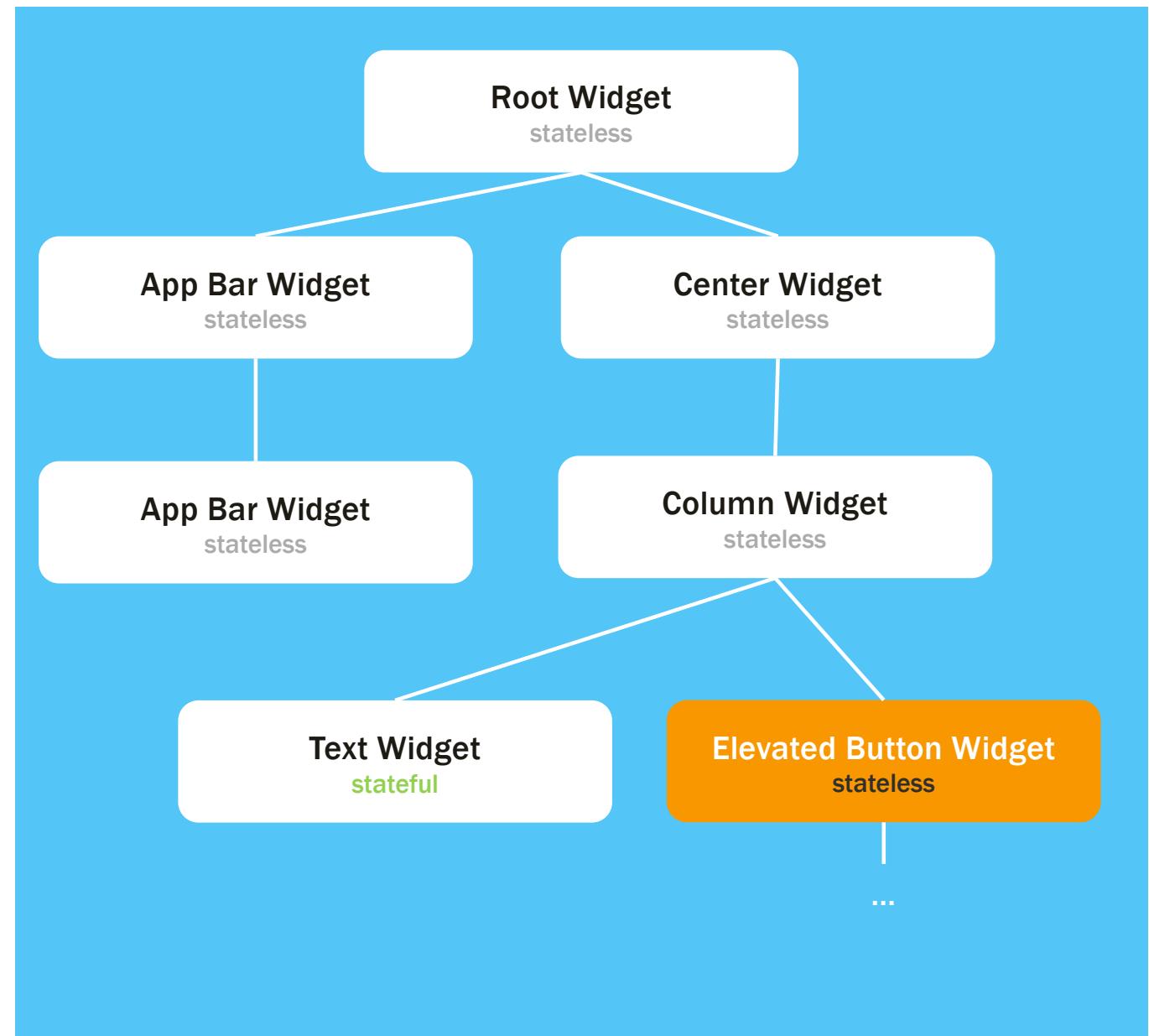
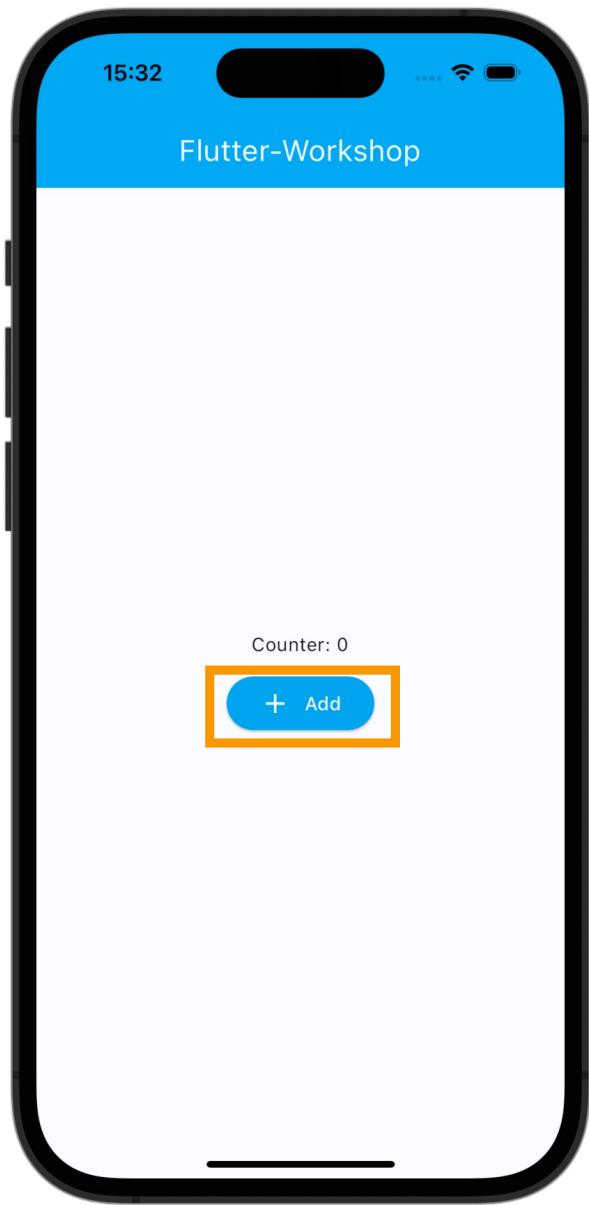


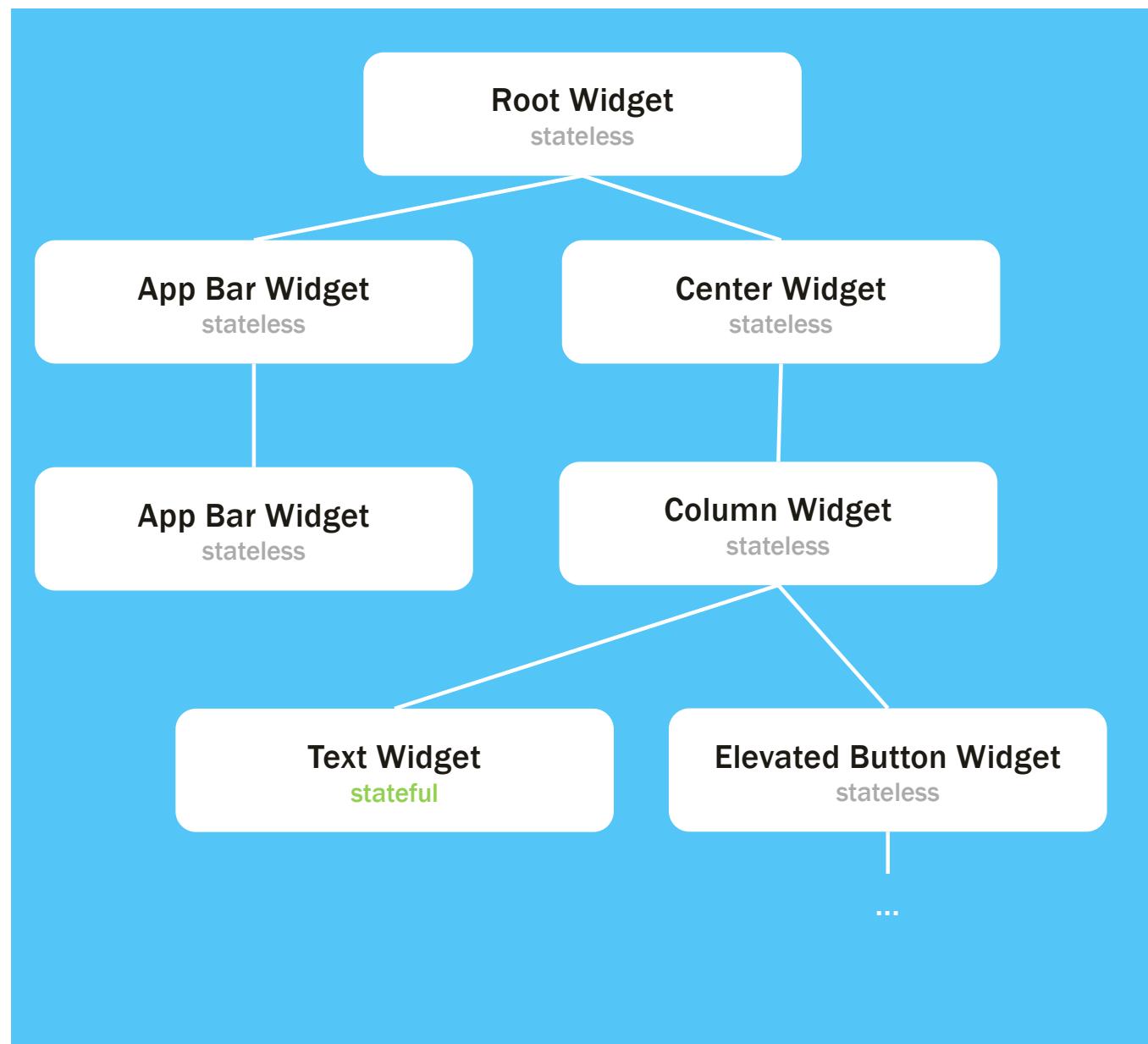
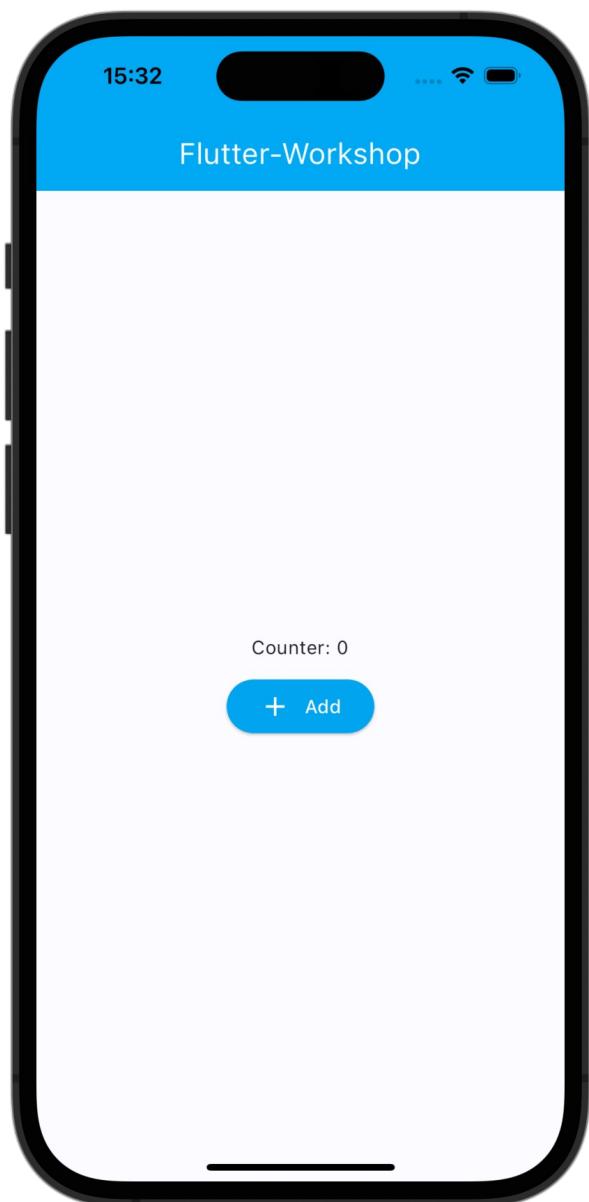










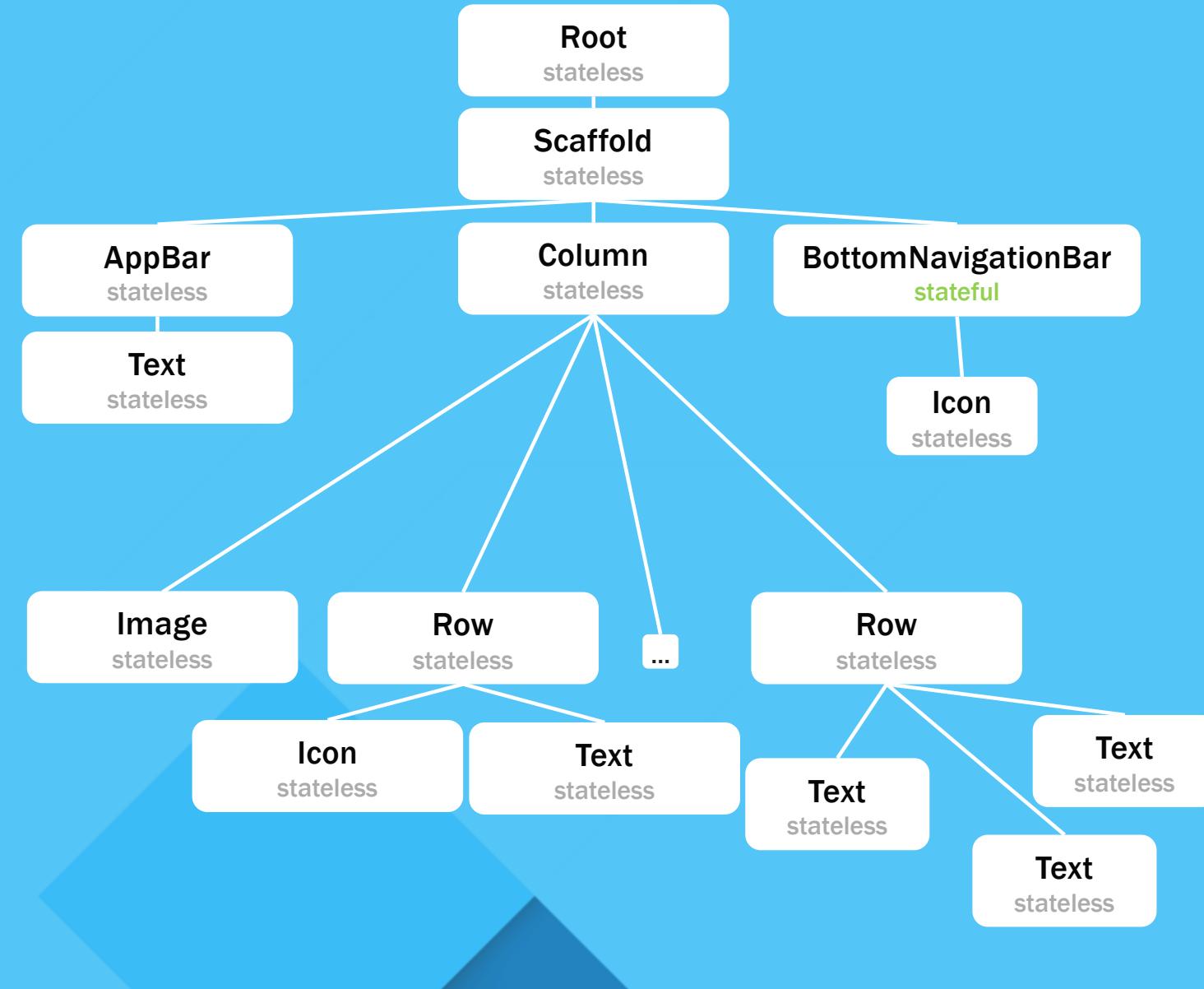




Flutter Basics

Now it's your turn to find the Widgets





Flutter Basics

What is BuildContext?

BuildContext

- Every widget has a unique BuildContext
- A reference or handle to the location of a widget within the widget tree
- Allows you to access data or functions from ancestor widgets

Summary



- **"Everything is a widget"**
- **A page is just a tree of widgets (including itself)**
- **There are stateless & stateful widgets**
- **Use BuildContext to navigate the tree**



Dart Crash-Course

Built-in types



Built-in types



```
int integer = 42;
```

Built-in types



```
int integer = 42;
```

```
String string = 'This is a string';
```

Built-in types



```
int integer = 42;  
  
String string = 'This is a string';  
  
bool boolean = true || false;
```

Built-in types



```
int integer = 42;
```

```
String string = 'This is a string';
```

```
bool boolean = true || false;
```

```
double d = 3.1415;
```

Built-in types



```
int integer = 42;
```

```
String string = 'This is a string';
```

```
bool boolean = true || false;
```

```
double d = 3.1415;
```

```
List<int> numbers = [1, 2, 3, 4, 5];
```

Further built-in types

- Record/Tuple
- Set
- Map
- Null
- Object (Everything except for Null)
- Void

Null safety



Null safety



```
int number = null;
```

Null safety



```
int number = null;
```



A value of type 'Null' can't be assigned to a variable of type 'int'.

Null safety



```
int? number = null;  
👉 Nullable type
```

Control flow



Control flow (conditional branching)



```
if (condition) {  
    // do something  
} else if (anotherCondition) {  
    // do something else  
} else {  
    // do something else  
}
```

Control flow (while loop)



```
while (!isDone()) {  
    doSomething();  
}
```

Functions



```
bool isDone() {  
    return true;  
}  
  
while (!isDone()) {  
    doSomething();  
}
```

Functions



```
void doSomething() {  
    print('Hello World!');  
}  
  
bool isDone() {  
    return true;  
}  
  
while (!isDone()) {  
    doSomething();  
}
```

Functions



```
bool isDone(int number) {  
    return number == 3;  
}  
  
int number = 0;  
while(!isDone(number)) {  
    doSomething();  
    number++;  
}
```

Anonymous functions



```
void doSomething(  
    void Function(int count) something,  
    int count,  
) {  
    something(count);  
}  
  
doSomething((count) {  
    print('Hello World! $count');  
}, number);
```

Anonymous functions



```
void doSomething(  
    void Function(int count) something,  
    int count,  
) {  
    something(count);  
}  
  
doSomething((count) { ↗ Functions are also objects  
    print('Hello World! $count');  
, number);
```

Control flow (for loop)



```
var message = StringBuffer('Dart is fun');
for (var i = 0; i < 5; i++) {
    message.write('!');
}
```

Control flow (enhanced for loop)



```
final participants = <Participant>[];
for (final participant in participants) {
    print('Hallo ${participant.name}!');
}
```

Classes



```
class Participant {  
    final String name;  
  
    Participant(this.name);  
}
```

👉 Positional parameter

Classes



```
class Participant {  
    final String name;  
  
    const Participant(this.name);  
}  
👉 All members can be known at  
compile time
```

Classes



```
class Participant {  
    final String name;  
    final int age;  
  
    const Participant(this.age, {  
        required this.name,  
    });    ✨ Required named  
}                                parameter
```

Classes



```
class Participant {  
    final String name;  
    final int age;  
    final String role;  
  
    const Participant(  
        this.age, {  
            required this.name,  
            this.role = 'member',  
        } );  
}      ➡️ Optional named parameter with  
          default value
```

Classes



```
class Participant {  
    final String name;  
    final int age;  
    final String role;  
    final String _password; ➔ Private member  
  
    const Participant(  
        this.age, {  
            required this.name,  
            this.role = 'member',  
            required String password,  
        }) : _password = password;  
}
```

Classes



```
class Participant {  
    final String name;  
    final int age;  
    final String role;  
    final String _password; ➡️ Private member  
  
    const Participant(  
        this.age, {  
            required this.name,  
            this.role = 'member',  
            required String password,  
        }) : _password = password;  
    } ➡️ Initializer
```

Classes



```
class Participant {  
    final String name;  
    final int age;  
    final String role;  
    final String _password;  
  
    const Participant(  
        this.age, {  
            required this.name,  
            this.role = 'member',  
            required String password,  
        }) : _password = password;  
  
    void greet() { ➡ Instance method  
        print('Hello $name!');  
    }  
}
```

Classes



```
class Participant {  
    final String name;  
    final int age;  
    final String role;  
    final String _password;  
  
    const Participant(  
        this.age, {  
            required this.name,  
            this.role = 'member',  
            required String password,  
        }) : _password = password;  
  
    bool get hasFun => true; ➡ Getter  
}
```

Classes



```
class Participant {  
    ...  
    String _password;  
    Participant(  
        this.age, {  
            required this.name,  
            this.role = 'member',  
            required String password,  
        }) : _password = password;  
  
    set password(String newPassword) { ➔ Setter  
        if (newPassword.length < 8) {  
            throw Exception('Password must be at least 8 characters  
long');  
        }  
        _password = newPassword;  
    }  
}
```

Classes



```
class Participant {  
    final String name;  
    final int age;  
    final String role;  
    String _password;  
    Participant(  
        this.age, {  
            required this.name,  
            this.role = 'member',  
            required String password,  
        }) : _password = password;  
  
    void greet() {  
        print('Hello $name!');  
    }  
    bool get hasFun => true;  
    set password(String newPassword) {  
        if (newPassword.length < 8) {  
            throw Exception('Password must be at least 8 characters long');  
        }  
        _password = newPassword;  
    }  
}
```

Classes



```
final participant = Participant(  
    19,  
    name: 'Julian Hartl',  
    password: 'mysecretpassword',  
);
```

Classes



```
final participant = Participant(  
Positional parameter👉 19,  
                                name: 'Julian Hartl',  
                                password: 'mysecretpassword',  
                                );
```

Classes



```
final participant = Participant(  
Positional parameter👉 19,  
                                name: 'Julian Hartl',  
                                password: 'mysecretpassword',👉 Named parameters  
);
```

Classes



```
final participant = Participant(  
Positional parameter👉 19,  
                                name: 'Julian Hartl',  
                                password: 'mysecretpassword',👉 Named parameters  
                                );  
participant.greet();👉 Call the greet method
```

Classes



```
final participant = Participant(  
  Positional parameter ↗ 19,  
  name: 'Julian Hartl',      ↗ Named parameters  
  password: 'mysecretpassword',  
);  
participant.greet(); ↗ Call the greet method  
Set new password ↗ participant.password = 'myverysecretpassword';
```

Classes



```
final participant = Participant(  
Positional parameter ↗ 19,  
                                name: 'Julian Hartl',  
                                password: 'mysecretpassword', ↗ Named parameters  
                                );  
participant.greet(); ↗ Call the greet method  
Set new password ↗ participant.password = 'myverysecretpassword';  
print(participant.hasFun()); ↗ Use hasFun getter
```

Summary

Me: I'm a dart developer
Friend: Cool, so you do flutter?



- Very Java/JavaScript like language
- Easy to pick up
- Lots of modern features

Shopping List App

Demo

Shopping List App

15:04

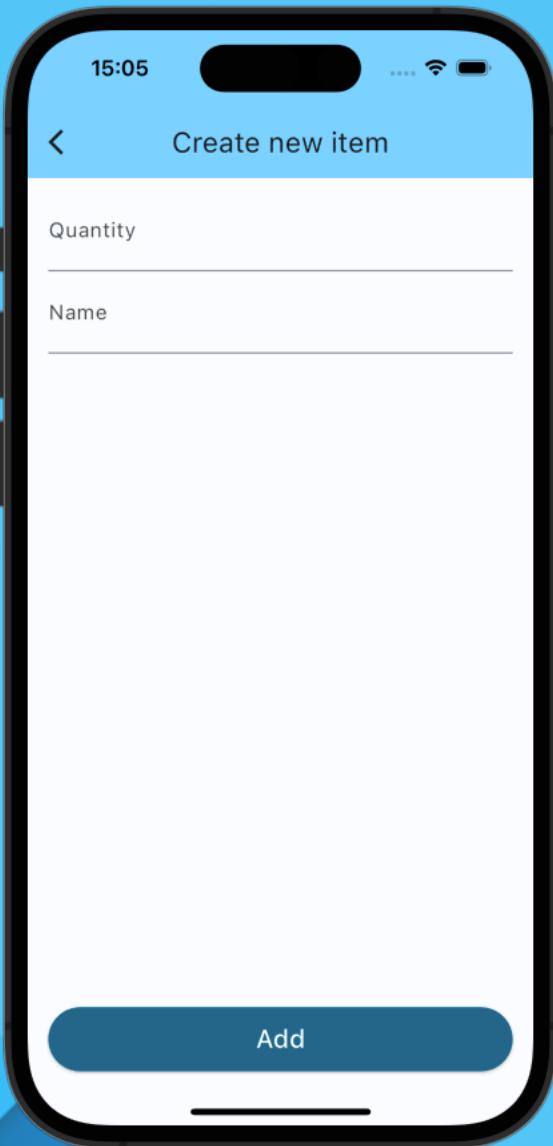
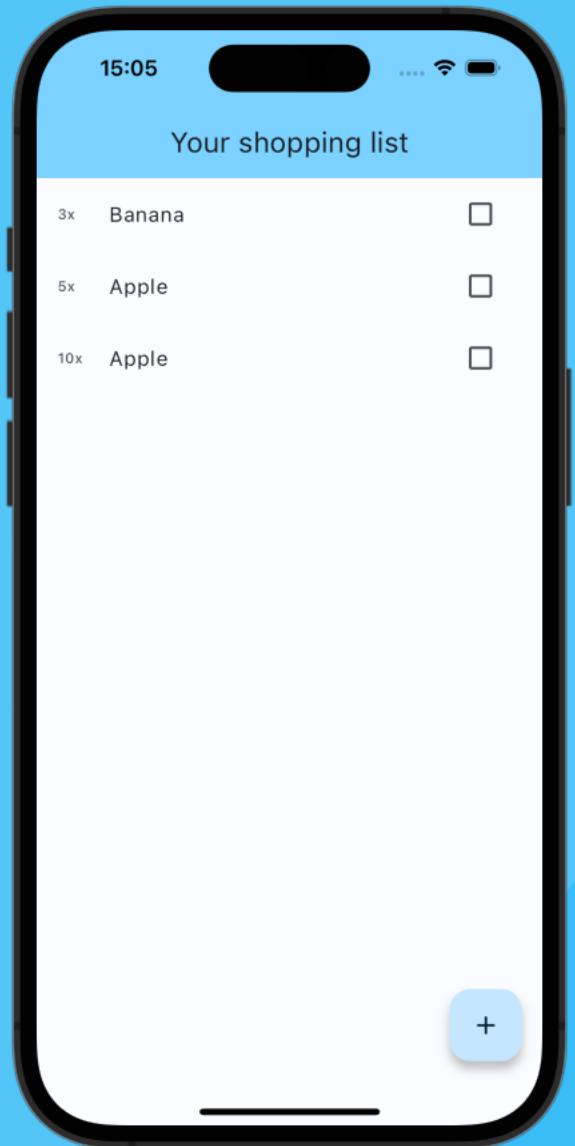


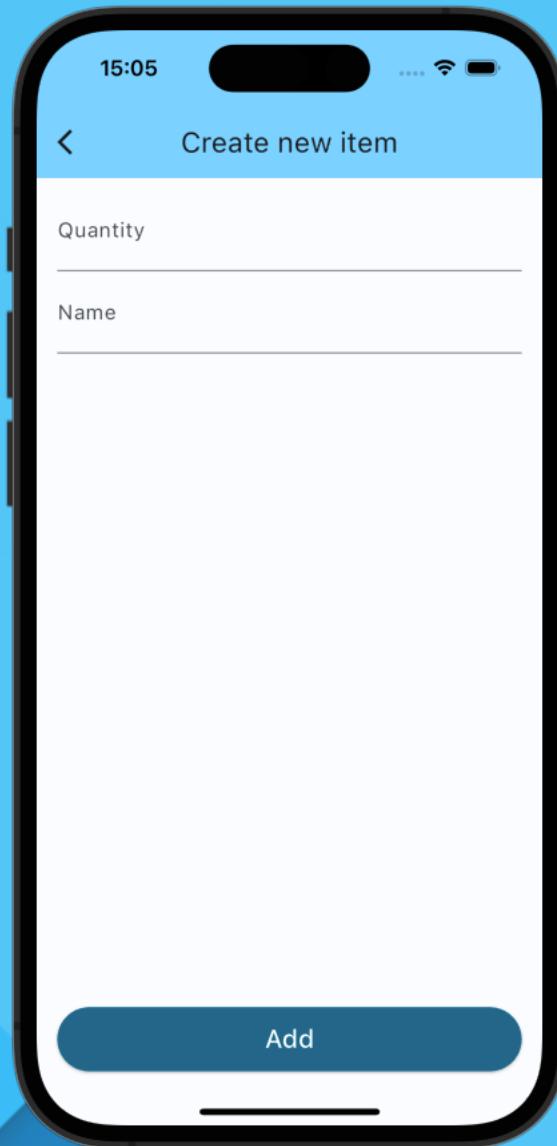
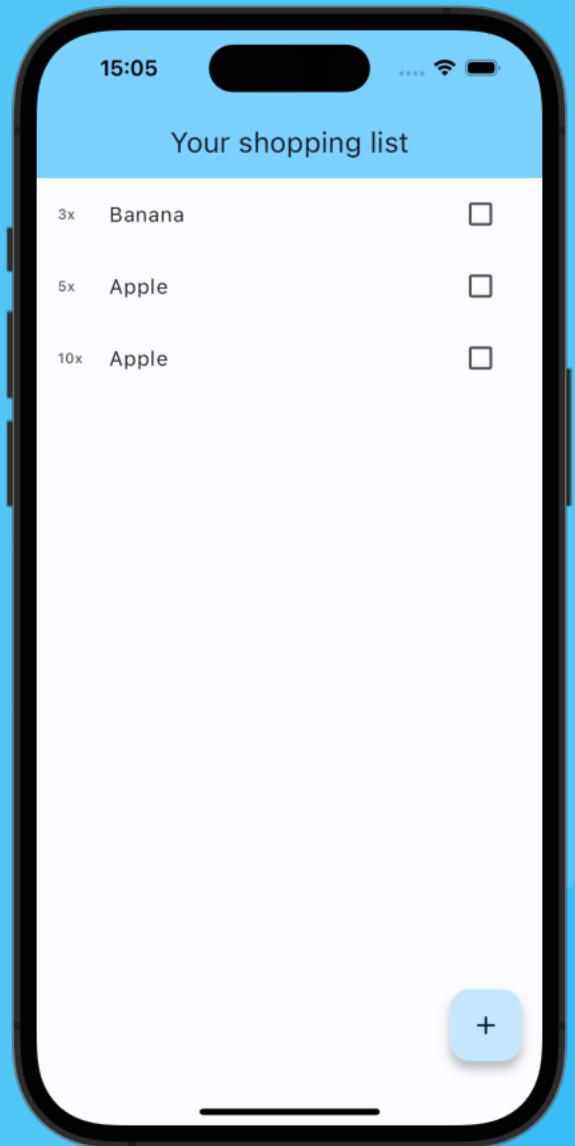
Your shopping list

3x Banana

5x Apple

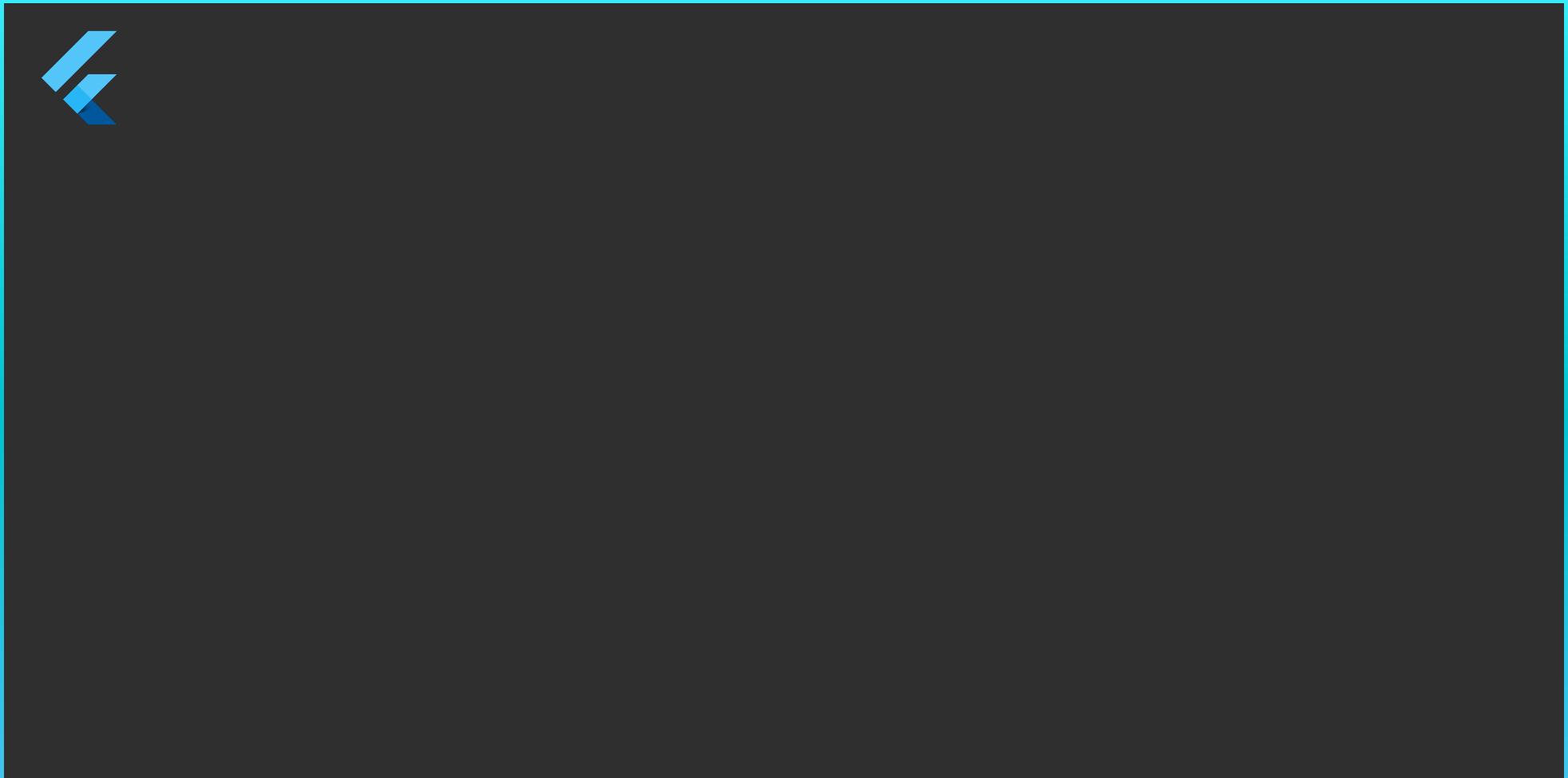
+

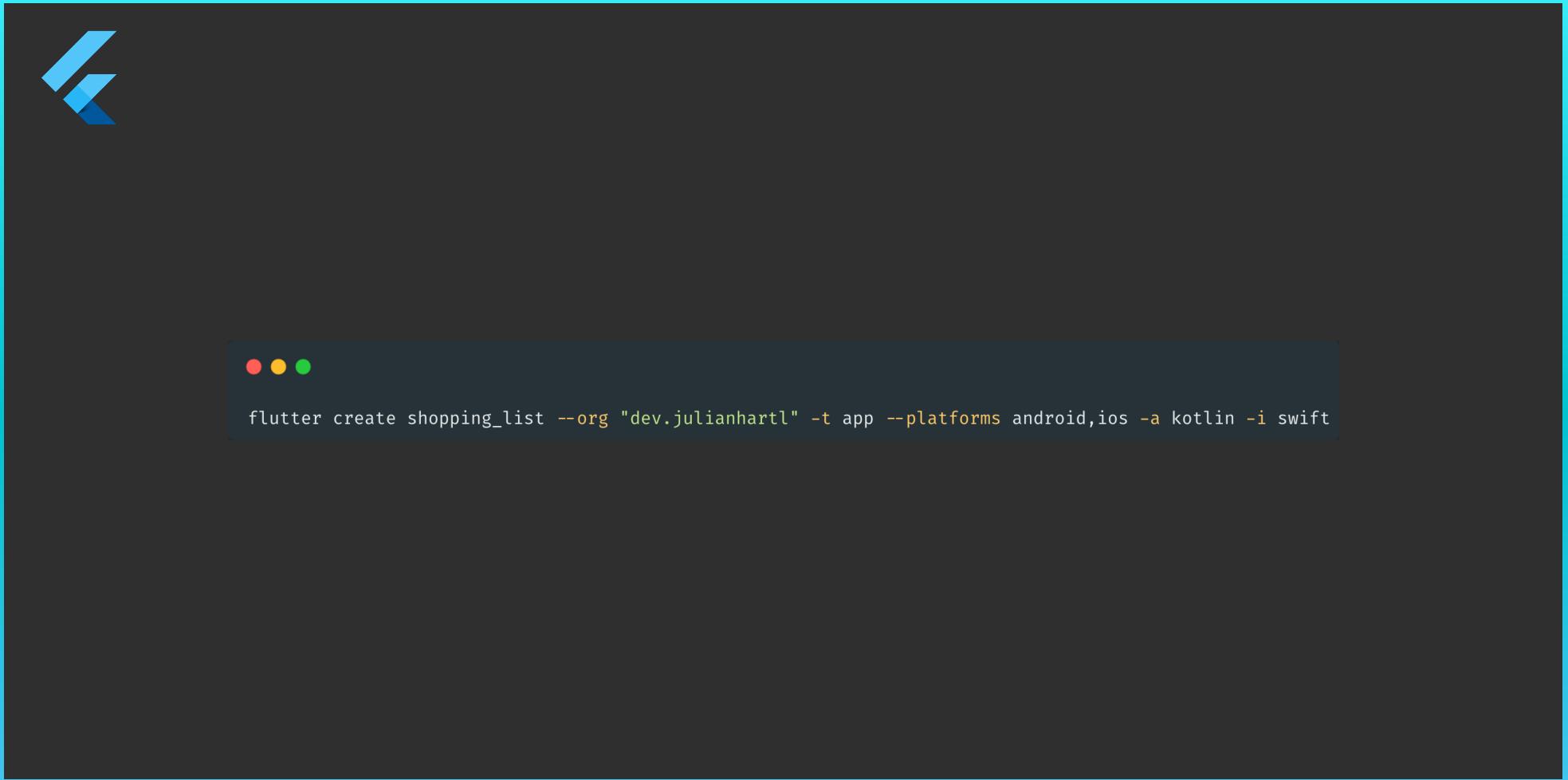


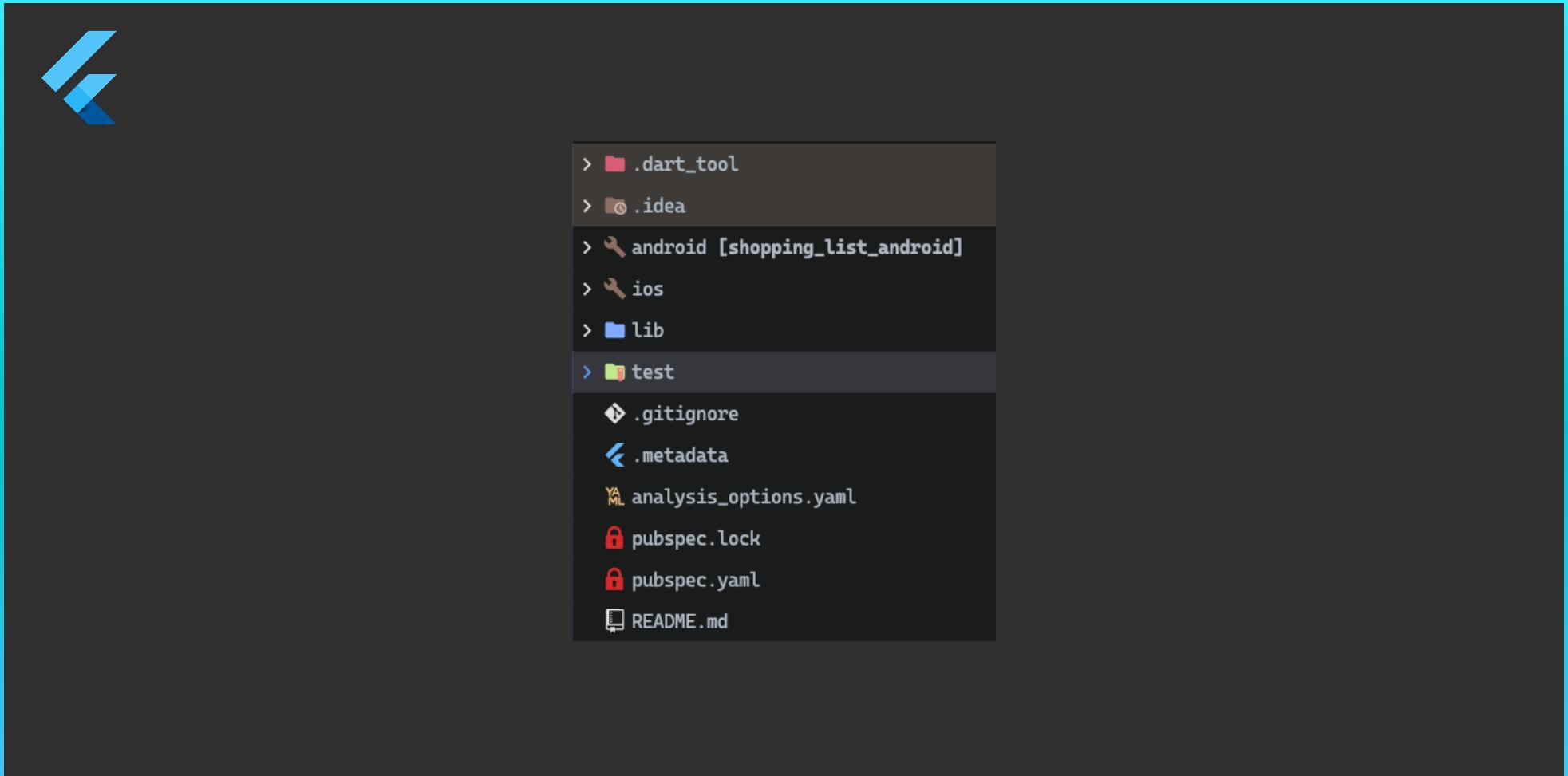


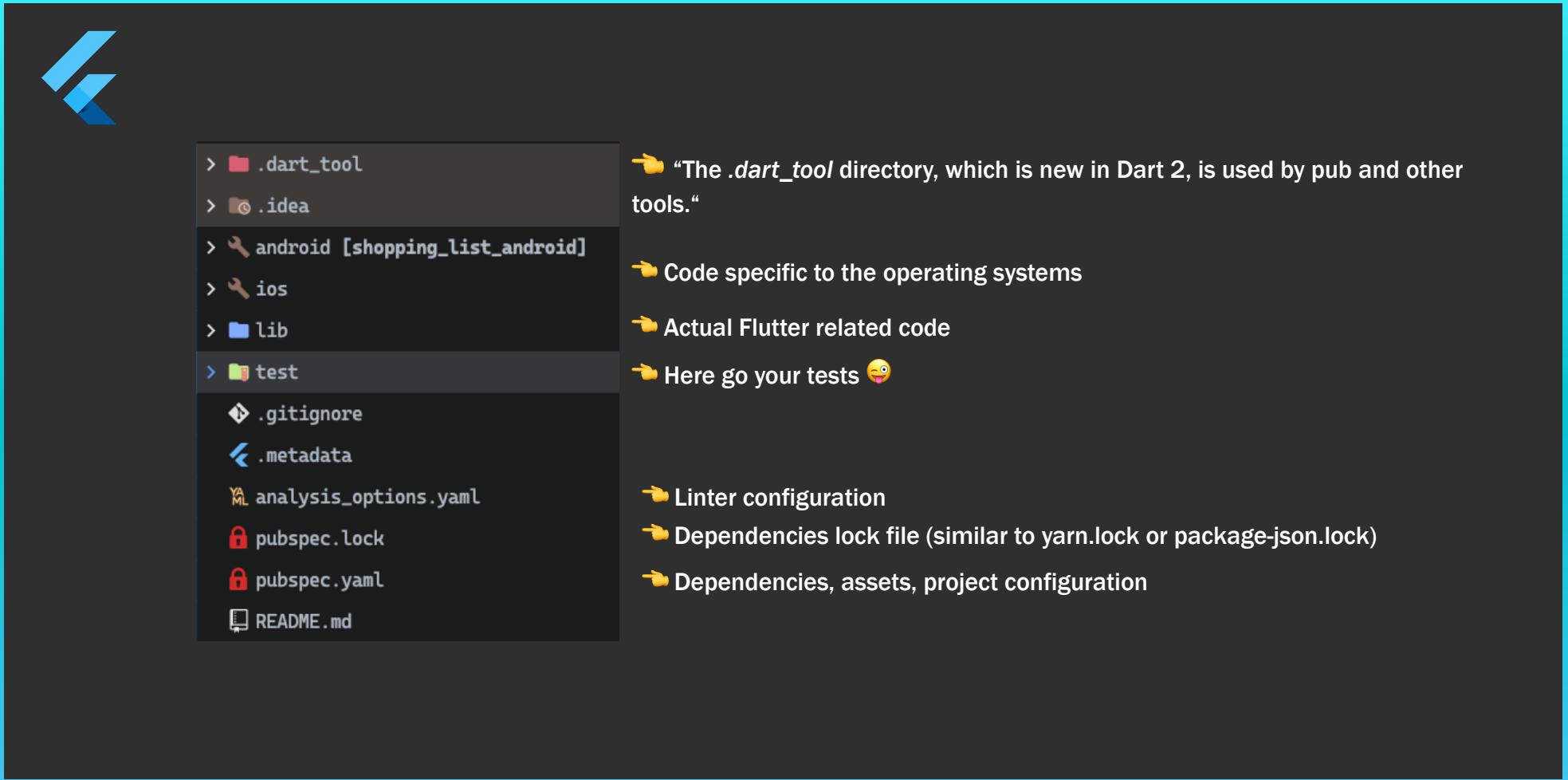
“Live”-Coding

Shopping List App









pubspec.yaml

Project configuration



```
name: shopping_list
description: A new Flutter project.
publish_to: 'none'

version: 1.0.0+1

environment:
  sdk: '>=3.1.3 <4.0.0'

dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.2

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^2.0.0
```

```
 name: shopping_list
description: A new Flutter project. ➡ Project configuration
publish_to: 'none'

version: 1.0.0+1

environment:
  sdk: '>=3.1.3 <4.0.0'

dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.2

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^2.0.0
```

```
 name: shopping_list
description: A new Flutter project. ➔ Project configuration
publish_to: 'none'

version: 1.0.0+1      ➔ Version + build number

environment:
  sdk: '>=3.1.3 <4.0.0'

dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.2

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^2.0.0
```

```
 name: shopping_list
description: A new Flutter project. ➔ Project configuration
publish_to: 'none'

version: 1.0.0+1      ➔ Version + build number

environment:
  sdk: '>=3.1.3 <4.0.0'    ➔ Supported dart versions

dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.2

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^2.0.0
```

```
 name: shopping_list
description: A new Flutter project. ➔ Project configuration
publish_to: 'none'

version: 1.0.0+1      ➔ Version + build number

environment:
  sdk: '>=3.1.3 <4.0.0'    ➔ Supported dart versions

dependencies:        ➔ Any packages which are included in the bundled app
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.2

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^2.0.0
```

```
 name: shopping_list
description: A new Flutter project. ➔ Project configuration
publish_to: 'none'

version: 1.0.0+1      ➔ Version + build number

environment:
  sdk: '>=3.1.3 <4.0.0'    ➔ Supported dart versions

dependencies:        ➔ Any packages which are included in the bundled app
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.2

dev_dependencies:   ➔ Any packages that are needed during development, but should not be
  flutter_test:     bundled in the final app
    sdk: flutter
  flutter_lints: ^2.0.0
```



```
flutter:  
  uses-material-design: true  
  assets:  
    - images/a_dot_burr.jpeg  
    - images/a_dot_ham.jpeg  
  fonts:  
    - family: Schyler  
      fonts:  
        - asset: fonts/Schyler-Regular.ttf  
        - asset: fonts/Schyler-Italic.ttf  
          style: italic  
    - family: Trajan Pro  
      fonts:  
        - asset: fonts/TrajanPro.ttf  
        - asset: fonts/TrajanPro_Bold.ttf  
          weight: 700
```



```
flutter:  
  uses-material-design: true  
  assets:  
    - images/a_dot_burr.jpeg  
    - images/a_dot_ham.jpeg  
  fonts:  
    - family: Schyler  
      fonts:  
        - asset: fonts/Schyler-Regular.ttf  
        - asset: fonts/Schyler-Italic.ttf  
          style: italic  
    - family: Trajan Pro  
      fonts:  
        - asset: fonts/TrajanPro.ttf  
        - asset: fonts/TrajanPro_Bold.ttf  
          weight: 700
```

👉 Any assets, that should be included in the bundled app



```
flutter:  
  uses-material-design: true  
  assets:  
    - images/a_dot_burr.jpeg  
    - images/a_dot_ham.jpeg      ➔ Any assets, that should be included in the bundled app  
  fonts:  
    - family: Schyler          ➔ Definition of customs fonts  
      fonts:  
        - asset: fonts/Schyler-Regular.ttf  
        - asset: fonts/Schyler-Italic.ttf  
          style: italic  
    - family: Trajan Pro  
      fonts:  
        - asset: fonts/TrajanPro.ttf  
        - asset: fonts/TrajanPro_Bold.ttf  
          weight: 700
```

Getting Started

Shopping List App



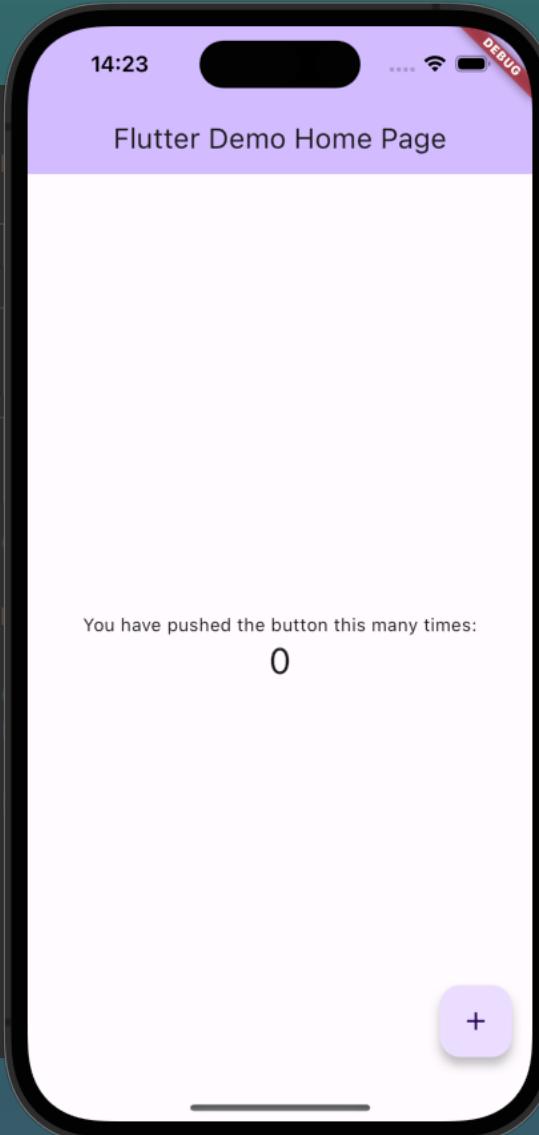
```
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
    const MyApp({super.key});

    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            title: 'Flutter Demo',
            theme: ThemeData(
                colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
                useMaterial3: true,
            ),
            home: const MyHomePage(title: 'Flutter Demo Home Page'),
        );
    }
}
```

```
import 'package:flutter/  
  
void main() {  
    runApp(const MyApp());  
}  
  
class MyApp extends StatelessWidget {  
    const MyApp({super.key});  
  
    @override  
    Widget build(BuildContext context) {  
        return MaterialApp(  
            title: 'Flutter Demo',  
            theme: ThemeData(  
                colorScheme: ColorScheme(  
                    useMaterial3: true),  
            ),  
            home: const MyHomePage(),  
        );  
    }  
}
```





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

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

class MyApp extends StatelessWidget {
    const MyApp({super.key});

    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            title: 'Flutter Demo',
            theme: ThemeData(
                colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
                useMaterial3: true,
            ),
            home: const MyHomePage(title: 'Flutter Demo Home Page'),
        );
    }
}
```



```
import 'package:flutter/material.dart'; ➔ Gives access to prebuilt widgets according to
void main() {
    runApp(const MyApp());
}

class MyApp extends StatelessWidget {
    const MyApp({super.key});

    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            title: 'Flutter Demo',
            theme: ThemeData(
                colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
                useMaterial3: true,
            ),
            home: const MyHomePage(title: 'Flutter Demo Home Page'),
        );
    }
}
```

Material-UI Guidelines



```
import 'package:flutter/material.dart'; ➔ Gives access to prebuilt widgets according to
void main() {                                         Material-UI Guidelines
    runApp(const MyApp()); ➔ Entry point of the application
}

class MyApp extends StatelessWidget {
    const MyApp({super.key});

    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            title: 'Flutter Demo',
            theme: ThemeData(
                colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
                useMaterial3: true,
            ),
            home: const MyHomePage(title: 'Flutter Demo Home Page'),
        );
    }
}
```



```
import 'package:flutter/material.dart'; ➔ Gives access to prebuilt widgets according to
void main() {                                         Material-UI Guidelines
    runApp(const MyApp()); ➔ Entry point of the application
}

class MyApp extends StatelessWidget { ➔ Extending from StatelessWidget to define a
    const MyApp({super.key});                         stateless widget

    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            title: 'Flutter Demo',
            theme: ThemeData(
                colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
                useMaterial3: true,
            ),
            home: const MyHomePage(title: 'Flutter Demo Home Page'),
        );
    }
}
```



```
import 'package:flutter/material.dart'; ➔ Gives access to prebuilt widgets according to
void main() {                                         Material-UI Guidelines
    runApp(const MyApp()); ➔ Entry point of the application
}

class MyApp extends StatelessWidget { ➔ Extending from StatelessWidget to define a
    const MyApp({super.key});                         stateless widget

    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            title: 'Flutter Demo', ➔ What is this?
            theme: ThemeData(
                colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
                useMaterial3: true,
            ),
            home: const MyHomePage(title: 'Flutter Demo Home Page'),
        );
    }
}
```

MaterialApp



“A convenience widget that wraps a number of widgets that are commonly required for Material Design applications. It builds upon a WidgetsApp by adding material-design specific functionality [...]”

Types of apps

MaterialApp

- Provides styles to material widgets according to the Material design language for iOS, Android and web
- Most commonly used

CupertinoApp

- Provides styles to cupertino widgets based on Apple's Human Interface Guidelines
- Used to build standard iOS-styled apps

WidgetsApp

- Basis for the other two apps
- Rarely to never used by actual Flutter developers



```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
```



```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData( ➔ Defines the theme of the application
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
```



```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData( ➡️ Defines the theme of the application
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
```



```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(
          seedColor: const Color(
            0xff38bdf8,    ➔ Custom hex color █ prefixed with 0xff
          ),
        ),
        useMaterial3: true,
      ),
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
```



```
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Shopping list',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(
          seedColor: const Color(
            0xff38bdf8,
          ),
        ),
        useMaterial3: true,
      ),
      home: const MyHomePage(title: 'Your shopping list'),
    );
  }
}
```

How do I run the app?

Using the flutter cli

Running the app

- Looks for any connected device (simulators, physical devices, ...)
- When started, type “r” to hot reload and “R“ to hot restart



14:38



Your shopping list

You have pushed the button this many times:

0



```
class MyApp extends StatelessWidget {  
  const MyApp({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      title: 'Shopping list',  
      theme: ThemeData(  
        colorScheme: ColorScheme.fromSeed(  
          seedColor: const Color(  
            0xff38bdf8,  
          ),  
          ),  
        useMaterial3: true,  
      ),  
      home: const MyHomePage(title: 'Your shopping list'),  
    );  
  }  
}
```

14:46

Your shopping list

You have pushed the button this many times:

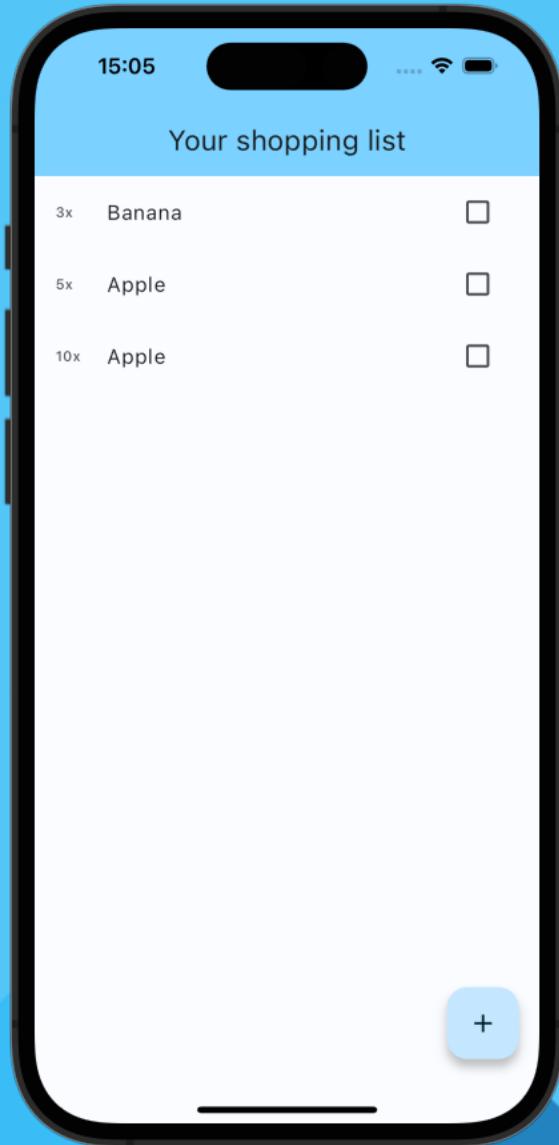
0



```
class MyApp extends StatelessWidget {  
  const MyApp({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      title: 'Shopping list',  
      debugShowCheckedModeBanner: false,  
      theme: ThemeData(  Hide DEBUG banner  
        colorScheme: ColorScheme.fromSeed(  
          seedColor: const Color(  
            0xff38bdf8,  
          ),  
          ),  
        useMaterial3: true,  
      ),  
      home: const MyHomePage(title: 'Your shopping list'),  
    );  
  }  
}
```

Creating the first page

Shopping List App



14:46

Your shopping list

You have pushed the button this many times:

0



```
class MyHomePage extends StatelessWidget {  
  const MyHomePage({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return const Placeholder();  
  }  
}
```

14:46

Your shopping list

You have pushed the button this many

0

No usages

```
class MyHomePage extends StatelessWidget {  
    const MyHomePage({super.key});  
  
    @override  
    Widget build(BuildContext context) {  
        return MaterialApp(  
            title: 'Shopping list',  
            debugShowCheckedModeBanner: false,  
            theme: ThemeData(  
                colorScheme: ColorScheme.fromSeed(  
                    seedColor: const Color(  
                        value: 0xff38bdf8,  
                    ), // Color  
                ), // ColorScheme.fromSeed  
                useMaterial3: true,  
            ), // ThemeData  
            home: const MyHomePage(title: 'Your shopping list'),  
        ); // MaterialApp  
    }  
}
```



14:46

Your shopping list

You have pushed the button this many times:

0



```
class MyHomePage extends StatelessWidget {  
    const MyHomePage({super.key});  
  
    @override  
    Widget build(BuildContext context) {  
        return const Placeholder();  
    }  
}
```

14:46

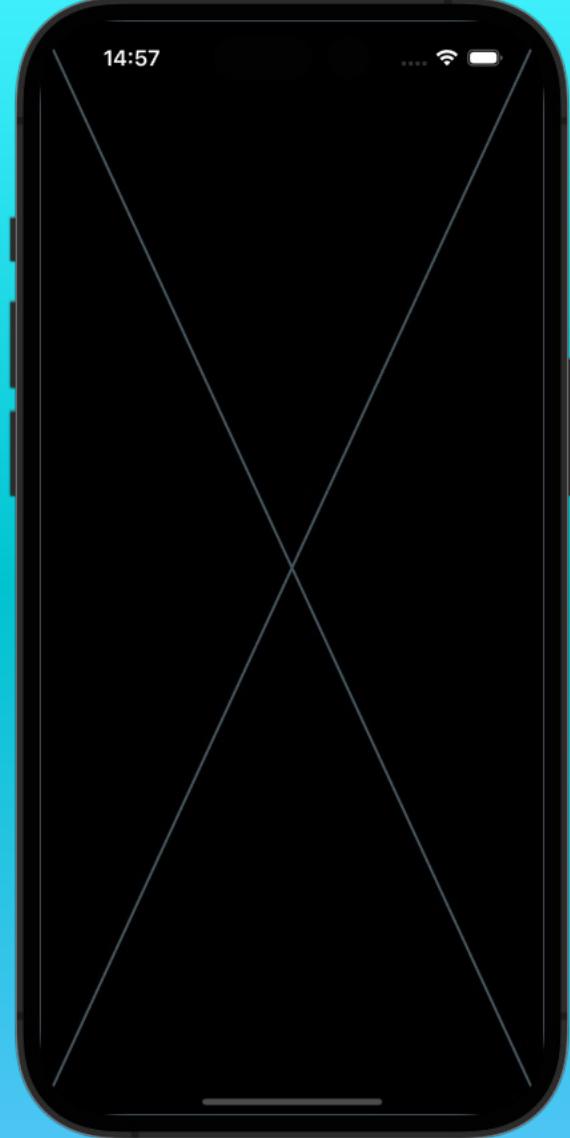
Your shopping list

You have pushed the button this many times:

0

+

```
class MyHomePage extends StatelessWidget {  
    const MyHomePage({super.key, required this.title});  
  
    final String title;  
  
    @override  
    Widget build(BuildContext context) {  
        return const Placeholder();  
    }  
}
```



14:57



```
class MyHomePage extends StatelessWidget {  
  const MyHomePage({super.key, required this.title});  
  
  final String title;  
  
  @override  
  Widget build(BuildContext context) {  
    return const Placeholder();  
  }  
}
```

Scaffold

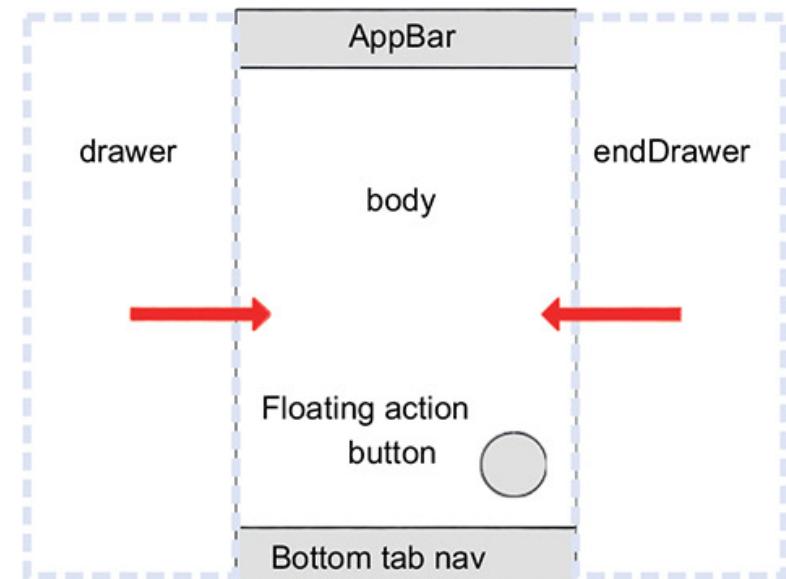


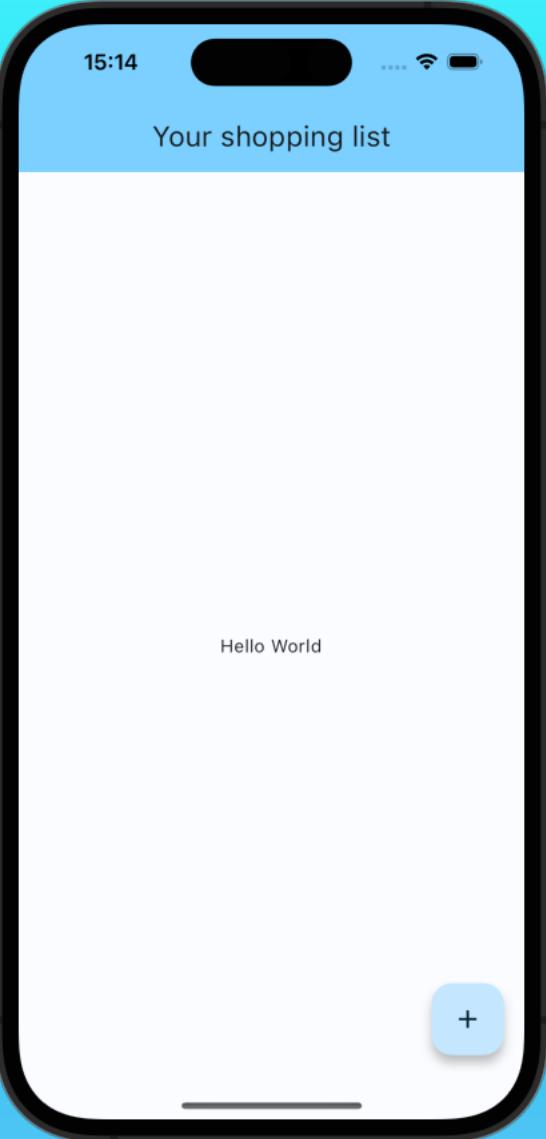
“Implements the basic Material Design visual layout structure.“

Scaffold



- **appBar:** “An app bar to display at the top of the scaffold.“
- **body:** “The primary content of the scaffold.“
- **floatingActionButton:** “A button displayed floating above body, in the bottom right corner.“
- **backgroundColor:** “The color of the Material widget that underlies the entire scaffold.“
- see all properties [here](#)





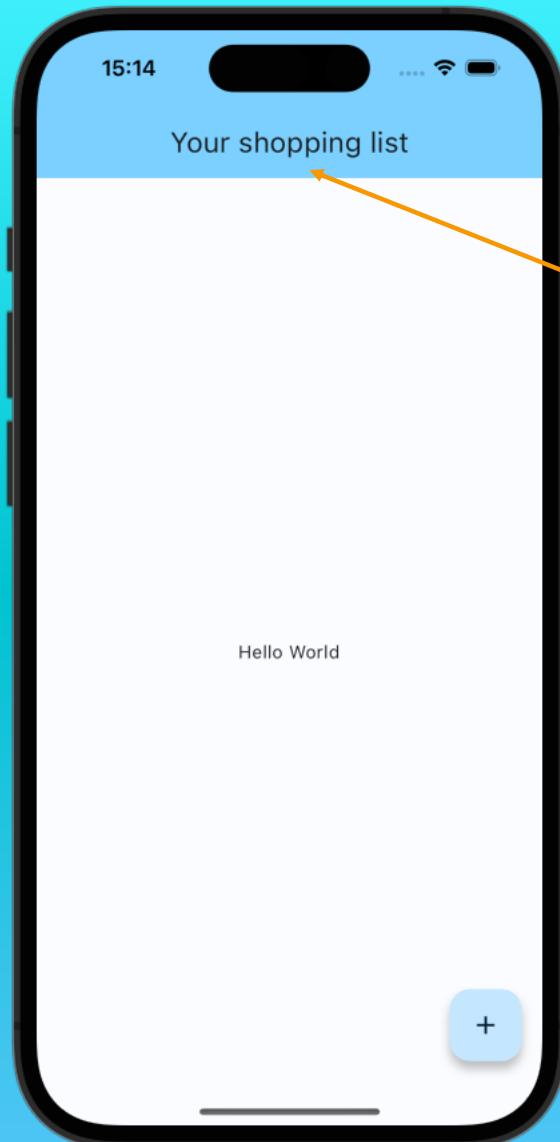
15:14

Your shopping list

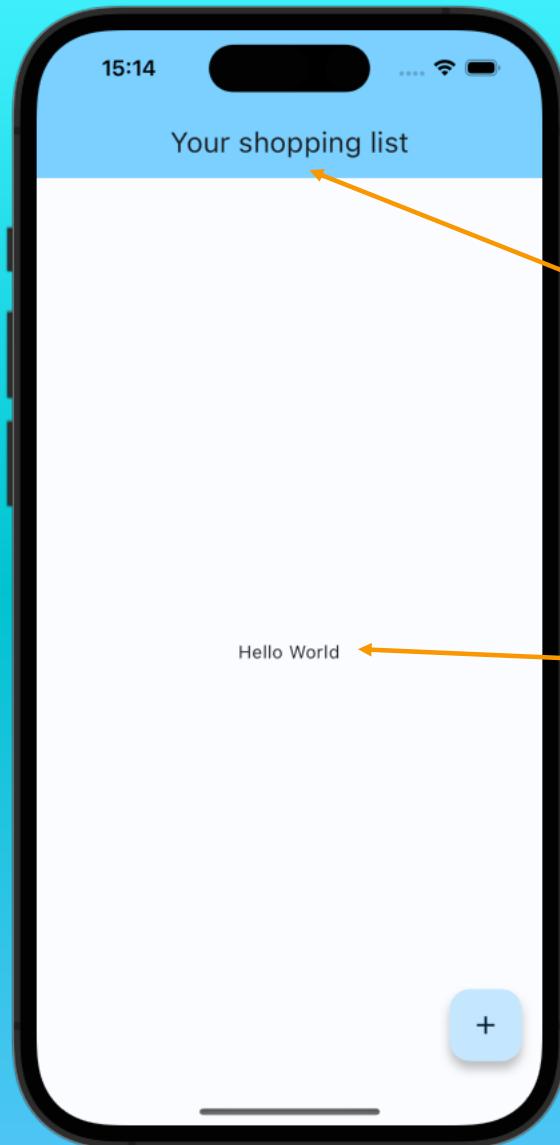
Hello World



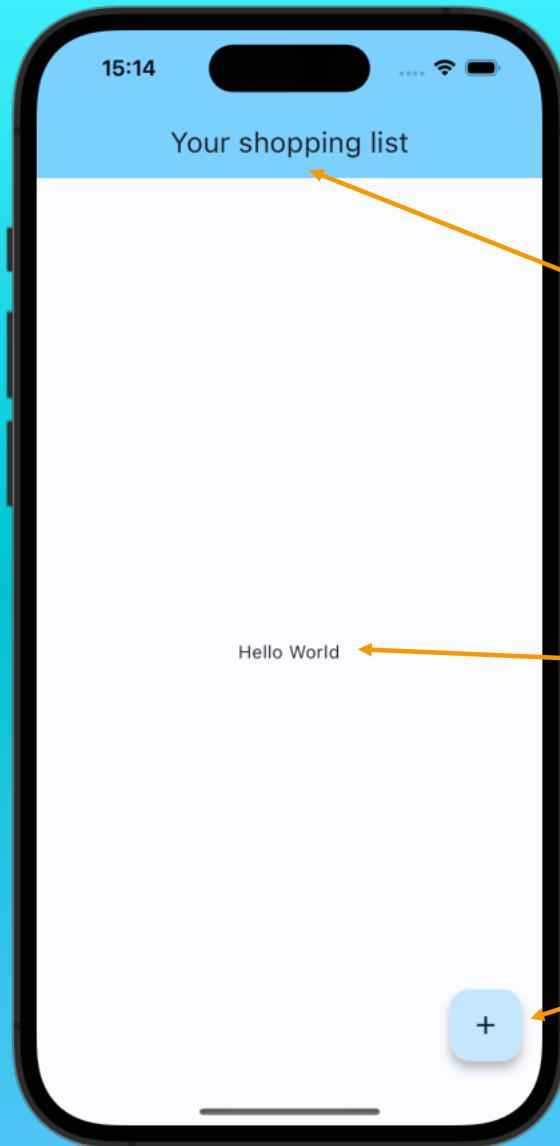
```
class MyHomePage extends StatelessWidget {  
    const MyHomePage({super.key, required this.title});  
  
    final String title;  
  
    @override  
    Widget build(BuildContext context) {  
        return Scaffold(  
            appBar: AppBar(  
                backgroundColor:  
                    Theme.of(context).colorScheme.inversePrimary,  
                title: Text(title),  
            ),  
            body: const Center(  
                child: Text('Hello World'),  
            ),  
            floatingActionButton: FloatingActionButton(  
                onPressed: () {},  
                child: const Icon(Icons.add),  
            ),  
        );  
    }  
}
```



```
class MyHomePage extends StatelessWidget {  
  const MyHomePage({super.key, required this.title});  
  
  final String title;  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        backgroundColor:  
          Theme.of(context).colorScheme.inversePrimary,  
        title: Text(title),  
      ),  
      body: const Center(  
        child: Text('Hello World'),  
      ),  
      floatingActionButton: FloatingActionButton(  
        onPressed: () {},  
        child: const Icon(Icons.add),  
      ),  
    );  
  }  
}
```



```
class MyHomePage extends StatelessWidget {  
    const MyHomePage({super.key, required this.title});  
  
    final String title;  
  
    @override  
    Widget build(BuildContext context) {  
        return Scaffold(  
            appBar: AppBar(  
                backgroundColor:  
                    Theme.of(context).colorScheme.inversePrimary,  
                title: Text(title),  
            ),  
            body: const Center(  
                child: Text('Hello World'),  
            ),  
            floatingActionButton: FloatingActionButton(  
                onPressed: () {},  
                child: const Icon(Icons.add),  
            ),  
        );  
    }  
}
```



```
class MyHomePage extends StatelessWidget {  
    const MyHomePage({super.key, required this.title});  
  
    final String title;  
  
    @override  
    Widget build(BuildContext context) {  
        return Scaffold(  
            appBar: AppBar(  
                backgroundColor:  
                    Theme.of(context).colorScheme.inversePrimary,  
                title: Text(title),  
            ),  
            body: const Center(  
                child: Text('Hello World'),  
            ),  
            floatingActionButton: FloatingActionButton(  
                onPressed: () {},  
                child: const Icon(Icons.add),  
            ),  
        );  
    }  
}
```

Center



“A widget that centers its child within itself.”

Center



“A widget that centers its child within itself.”



AppBar

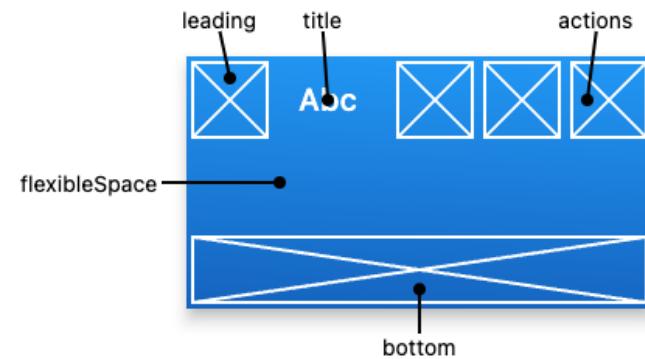


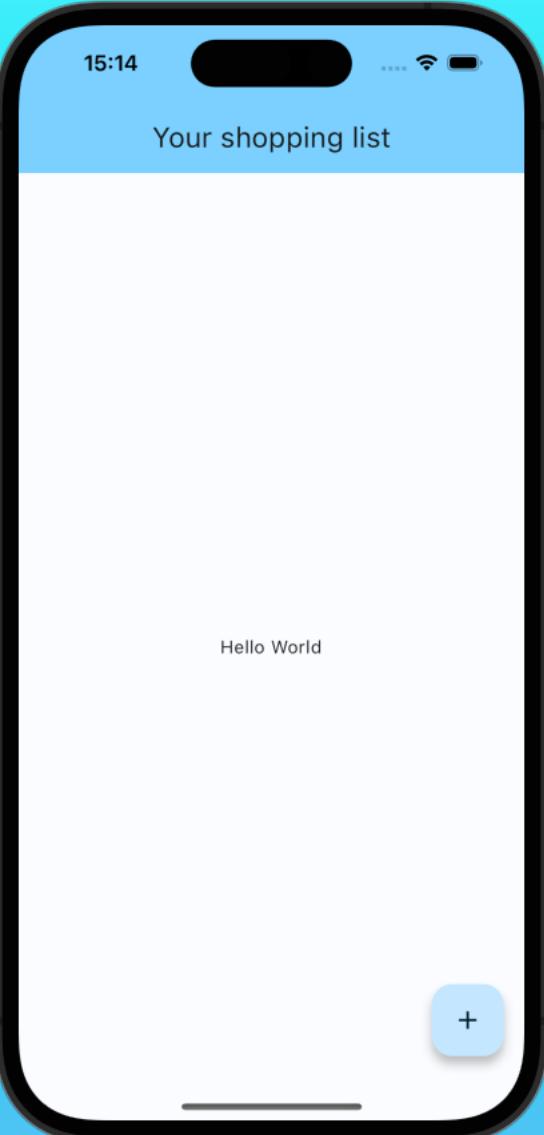
“A Material Design app bar.“

AppBar



- **title:** “The primary widget displayed in the app bar.”
- **actions:** “A list of Widgets to display in a row after the title widget.”
- **leading:** “A widget to display before the toolbar’s title.”
- **bottom:** “This widget appears across the bottom of the app bar.”
- **see all properties [here](#)**





15:14

Your shopping list

Hello World



```
class MyHomePage extends StatelessWidget {  
    const MyHomePage({super.key, required this.title});  
  
    final String title;  
  
    @override  
    Widget build(BuildContext context) {  
        return Scaffold(  
            appBar: AppBar(  
                backgroundColor:  
                    Theme.of(context).colorScheme.inversePrimary,  
                title: Text(title),  
            ),  
            body: const Center(  
                child: Text('Hello World'),  
            ),  
            floatingActionButton: FloatingActionButton(  
                onPressed: () {},  
                child: const Icon(Icons.add),  
            ),  
        );  
    }  
}
```

15:21

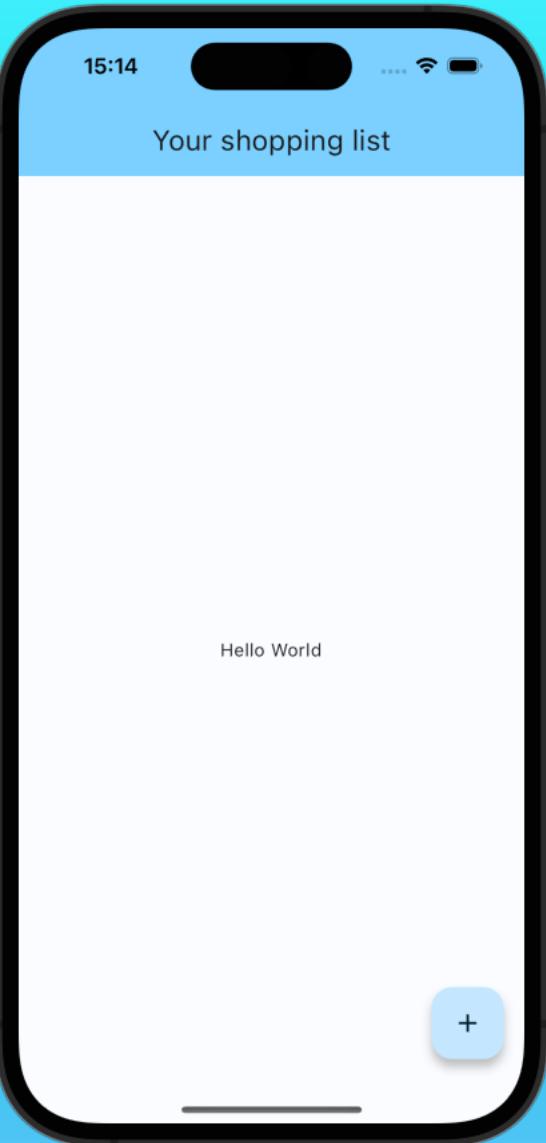
Your shopping list

Hello World



```
@override  
Widget build(BuildContext context) {  
  return Scaffold(  
    appBar: AppBar(  
      backgroundColor:  
        Theme.of(context).colorScheme.inversePrimary,  
      title: Text(title),  
    ),  
    body: const Center(  
      child: Text('Hello World'),  
    ),  
    floatingActionButton: const Text('1'),  
  );  
}
```

👉 floatingActionButton is not of type
FloatingActionButton, but Widget



15:14

Your shopping list

Hello World



```
@override  
Widget build(BuildContext context) {  
  return Scaffold(  
    appBar: AppBar(  
      backgroundColor:  
        Theme.of(context).colorScheme.inversePrimary,  
      title: Text(title),  
    ),  
    body: const Center(  
      child: Text('Hello World'),  
    ),  
    floatingActionButton: FloatingActionButton(  
      onPressed: () {},  
      child: const Icon(Icons.add),  
    ),  
  );  
}
```

Defining the models

Shopping List App

15:14

Your shopping list

Hello World



```
class ShoppingList {  
    final List<ShoppingListItem> items;  
  
    ShoppingList() : items = [];  
}
```

15:14

Your shopping list

Hello World



```
class ShoppingList {  
    final List<ShoppingListItem> items;  
  
    ShoppingList() : items = [];  
}  
  
class ShoppingListItem {  
    final int id;  
    final String name;  
    final int quantity;  
    bool isBought;  
  
    ShoppingListItem({  
        required this.id,  
        required this.name,  
        required this.quantity,  
        required this.isBought,  
    });  
}
```

15:14

Your shopping list

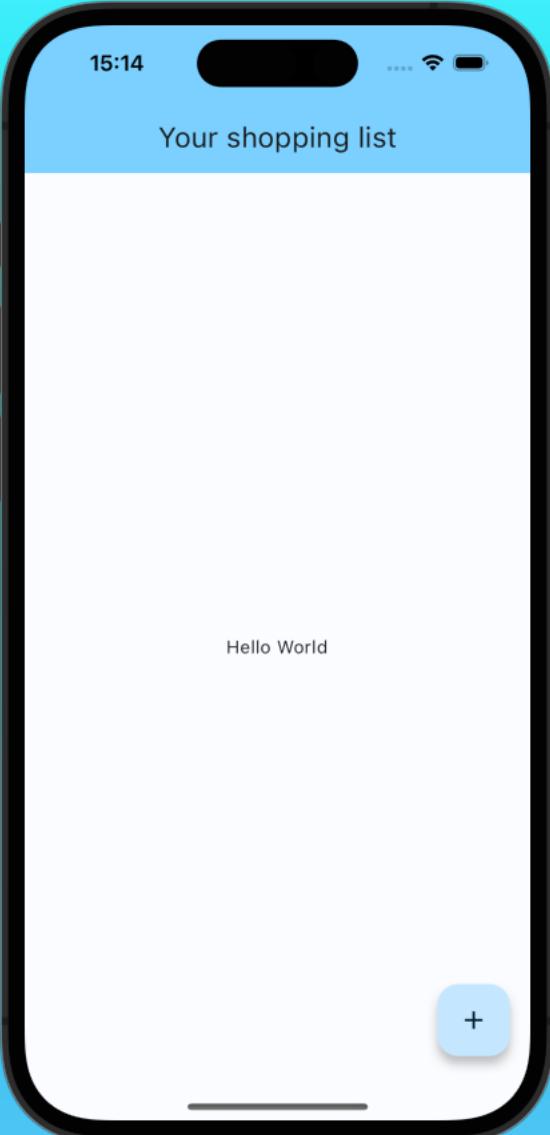
Hello World



```
class ShoppingList {  
    final List<ShoppingListItem> items;  
  
    static int _nextId = 0;  
  
    ShoppingList() : items = [];  
  
    void addItem({  
        required String name,  
        required int quantity,  
    }) {  
        items.add(  
            ShoppingListItem(  
                id: _nextId++,  
                name: name,  
                quantity: quantity,  
                isBought: false,  
            ),  
        );  
    }  
}
```

List of ShoppingListItems

Shopping List App



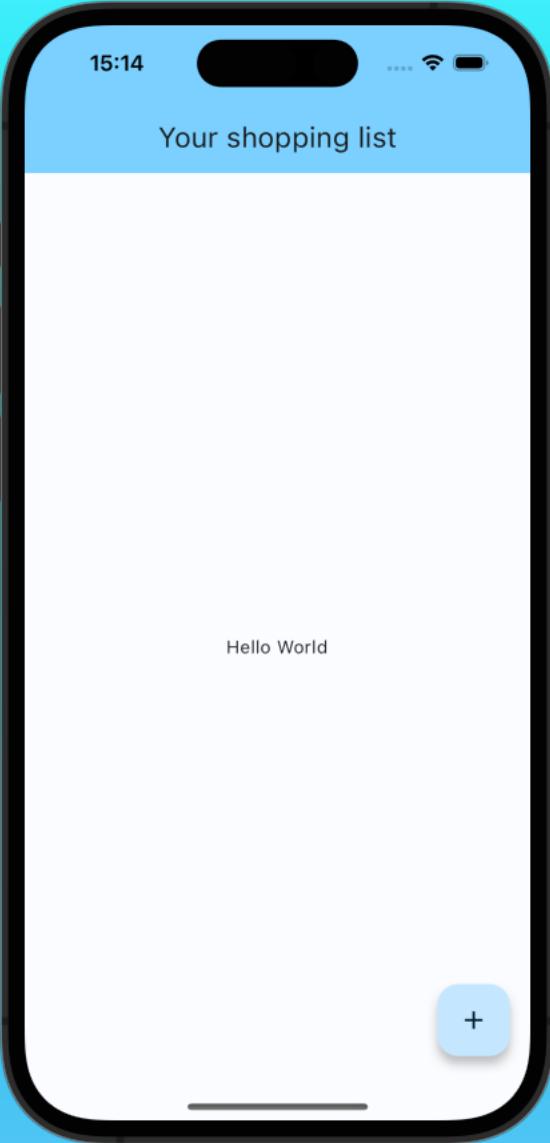
15:14

Your shopping list

Hello World



```
class ShoppingListView extends StatefulWidget {  
  const ShoppingListView({super.key});  
  
  @override  
  State<ShoppingListView> createState() =>  
  _ShoppingListViewState();  
}  
  
class _ShoppingListViewState extends State<ShoppingListView> {  
  @override  
  Widget build(BuildContext context) {  
    return const Placeholder();  
  }  
}
```

A black iPhone X is shown on the left side of the image. The screen displays a mobile application titled "Your shopping list". Below the title, the text "Hello World" is visible. At the bottom right of the screen is a blue circular button with a white plus sign (+). The phone is set against a light blue background.

15:14

Your shopping list



```
class _ShoppingListViewState extends State<ShoppingListView>
{
    ShoppingList _shoppingList = ShoppingList();

    @override
    Widget build(BuildContext context) {
        return const Placeholder();
    }
}
```

ListView

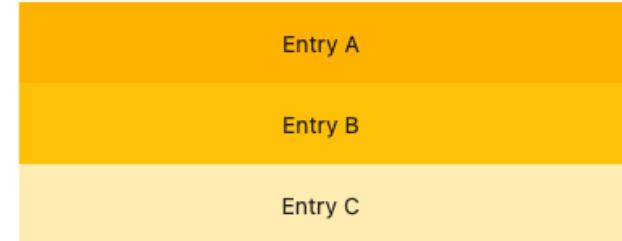


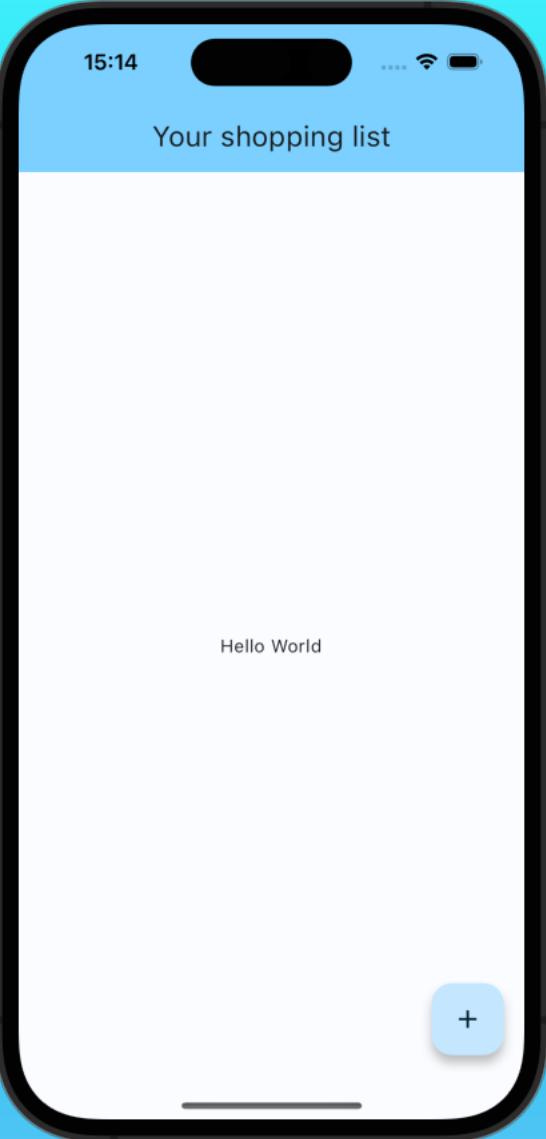
“A scrollable list of widgets arranged linearly.“

ListView



- **children:** The widgets layed out along the main axis.
- **scrollDirection:** “The axis along which the scroll view’s offset increases.”
- **shrinkWrap:** “Whether the extent of the scroll view in the scrollDirection should be determined by the contents being viewed.”
- **controller:** “An object that can be used to control the position to which this scroll view is scrolled.”
- **see all properties [here](#)**



A black iPhone X is shown on the left side of the image. The screen displays a mobile application titled "Your shopping list". Below the title, the text "Hello World" is visible. At the bottom right of the screen is a blue circular button with a white plus sign (+). The phone is set against a light blue background.

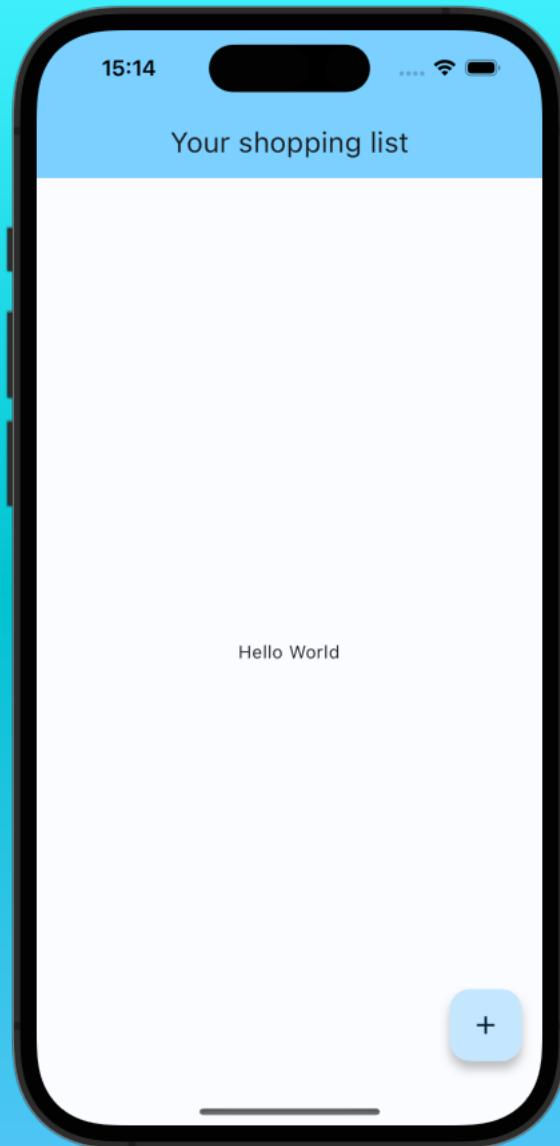
15:14

Your shopping list



```
class _ShoppingListViewState extends State<ShoppingListView>
{
    ShoppingList _shoppingList = ShoppingList();

    @override
    Widget build(BuildContext context) {
        return ListView(
            children: [],
        );
    }
}
```



ListTile



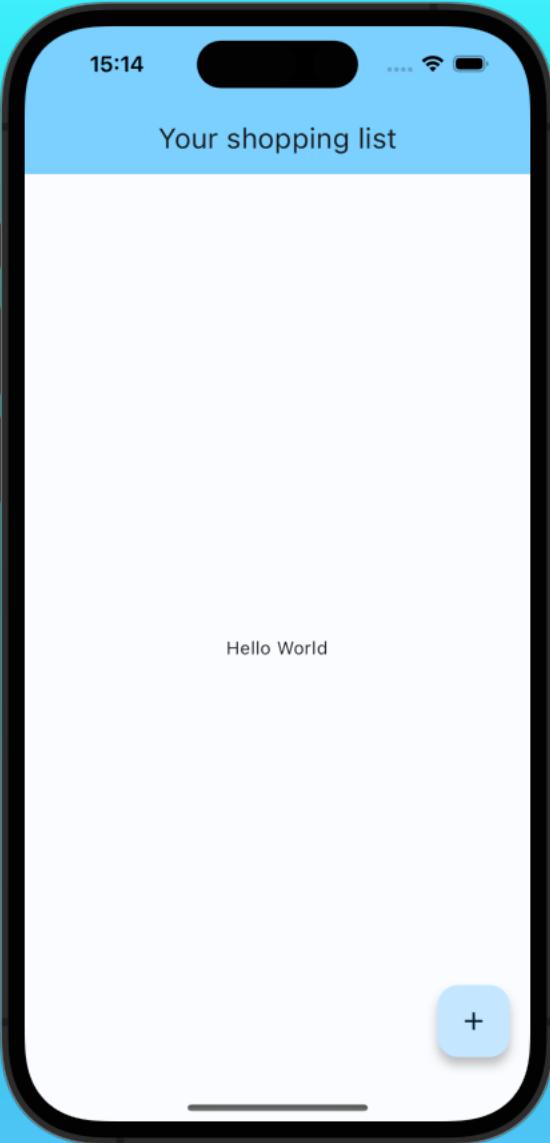
“A single fixed-height row that typically contains some text as well as a leading or trailing icon.”

ListTile



- **title:** “The primary content of the list tile.”
- **leading:** “A widget to display before the title.”
- **trailing:** “A widget to display after the title.”
- **subtitle:** “Additional content displayed below the title.”
- **onTap:** “Called when the user taps this list tile.”
- **see all properties [here](#)**

	One-line ListTile	⋮
	One-line with leading widget	⋮
	One-line with trailing widget	⋮
	One-line with both widgets	⋮
One-line dense ListTile		
	Two-line ListTile Here is a second line	⋮
	Three-line ListTile A sufficiently long subtitle warrants three lines.	⋮



15:14

Your shopping list

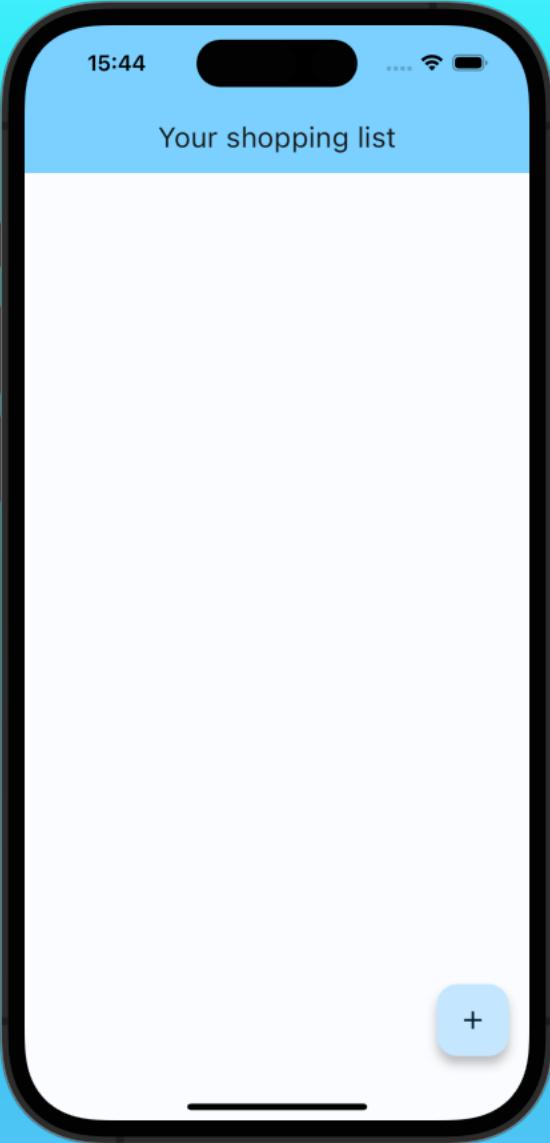
Hello World

+



```
class _ShoppingListViewState extends State<ShoppingListView> {
  ShoppingList _shoppingList = ShoppingList();

  @override
  Widget build(BuildContext context) {
    return ListView(
      children: [
        for (final item in _shoppingList.items)
          ListTile(
            leading: Text(item.id.toString()),
            title: Text(item.name),
          )
      ],
    );
  }
}
```

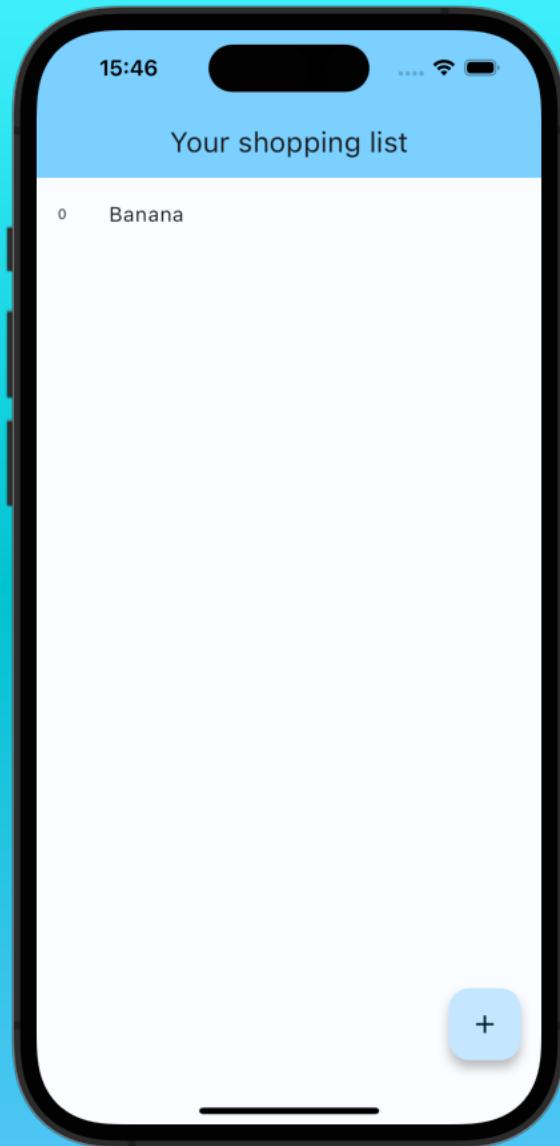
A black iPhone X is shown on the left side of the image. The screen displays a mobile application titled "Your shopping list". The interface includes a blue header bar with the title, a white main content area, and a floating action button at the bottom right containing a plus sign. The phone is set against a light blue background.

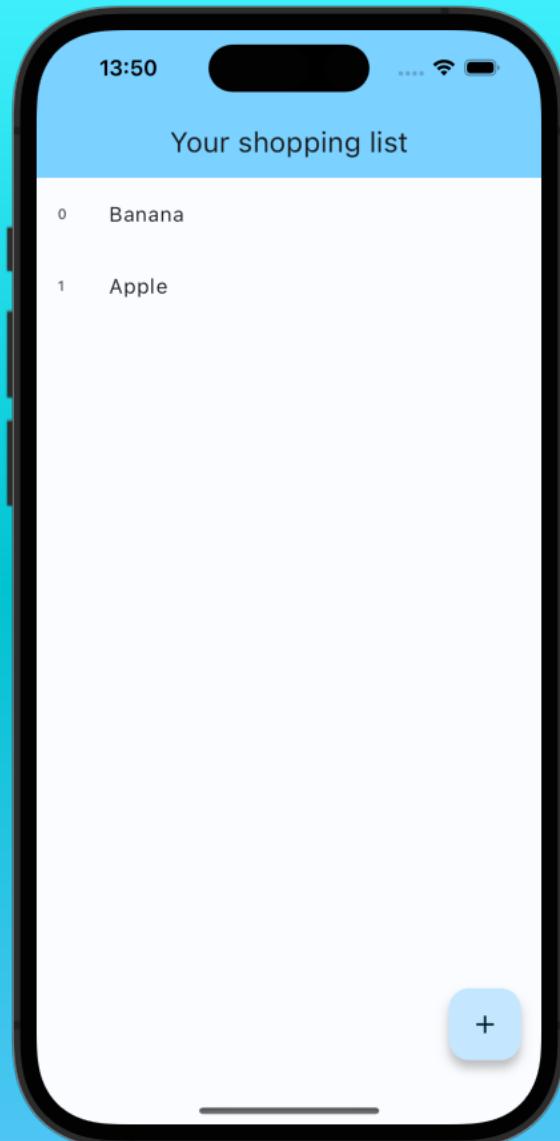
15:44

Your shopping list



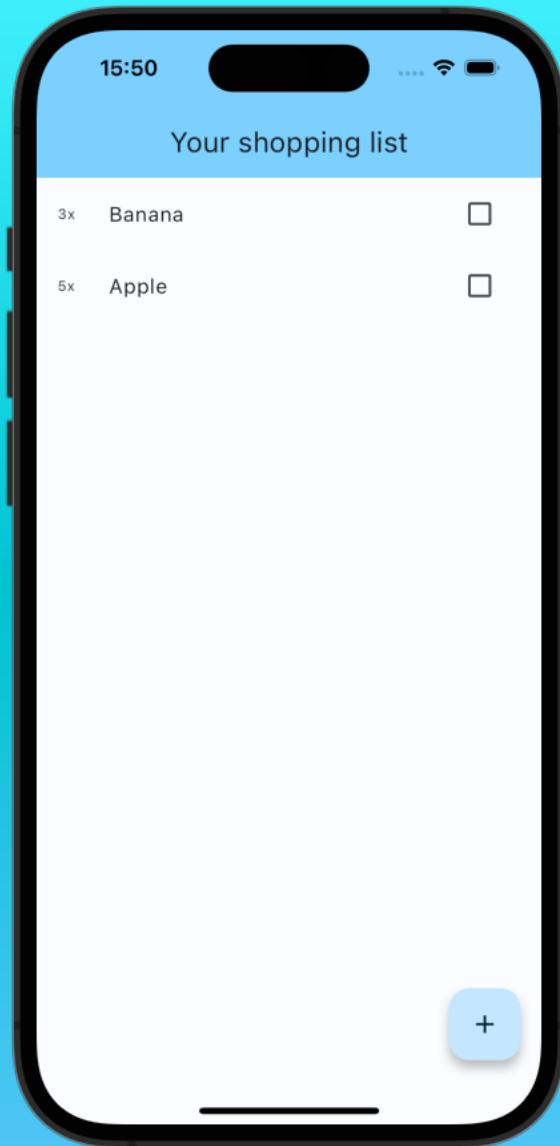
```
class MyHomePage extends StatelessWidget {  
  const MyHomePage({super.key, required this.title});  
  
  final String title;  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        backgroundColor:  
          Theme.of(context).colorScheme.inversePrimary,  
        title: Text(title),  
      ),  
      body: const ShoppingListView(),  
      floatingActionButton: FloatingActionButton(  
        onPressed: () {},  
        child: const Icon(Icons.add),  
      ),  
    );  
  }  
}
```





```
class _ShoppingListViewState extends State<ShoppingListView> {
  ShoppingList _shoppingList = ShoppingList()
    ..addItem(
      name: 'Banana',
      quantity: 3,
    )
    ..addItem(
      name: 'Apple',
      quantity: 5);

  @override
  Widget build(BuildContext context) {
    return ListView(
      children: [
        for (final item in _shoppingList.items)
          ListTile(
            leading: Text(item.id.toString()),
            title: Text(item.name),
          )
      ],
    );
}
```



CheckBox



“A Material Design checkbox.“

CheckBox



- **value:** “Whether this checkbox is checked.”
- **onChanged:** “Called when the value of the checkbox should change.”
- **see all properties [here](#)**

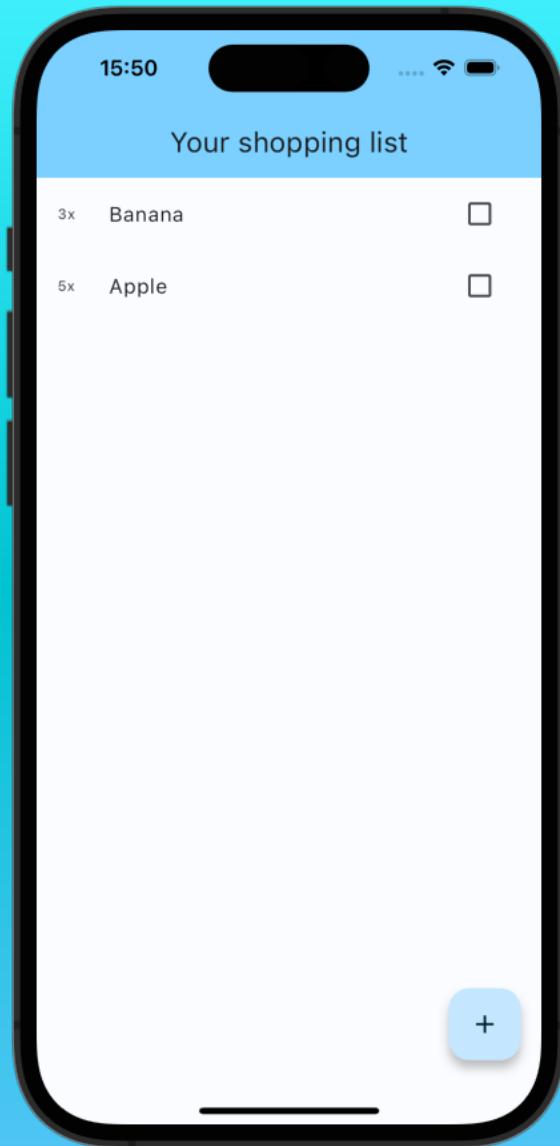
Checkbox 1

Checkbox 2

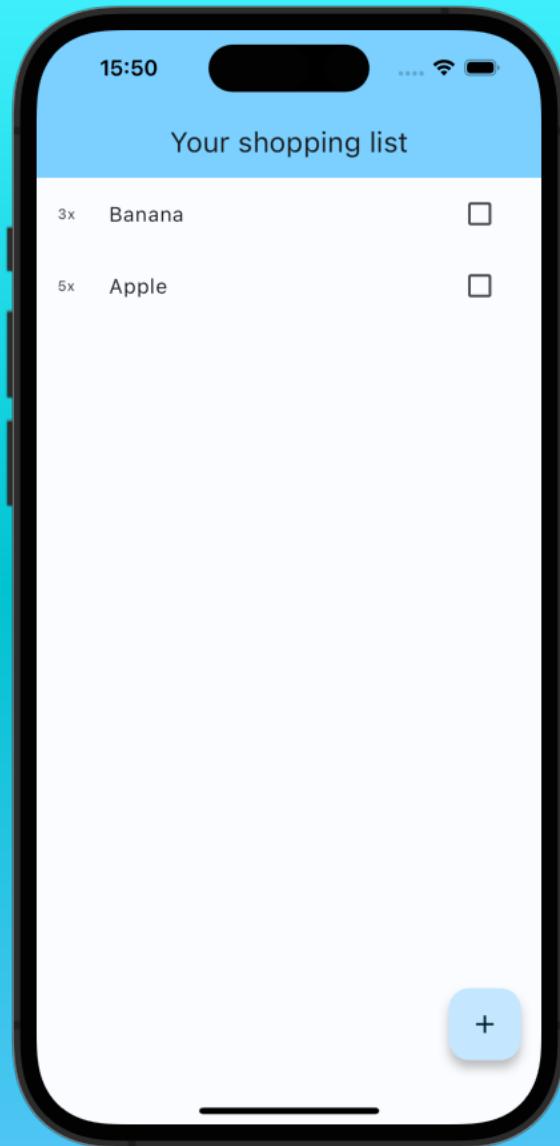
Checkbox 3

Checkbox 4

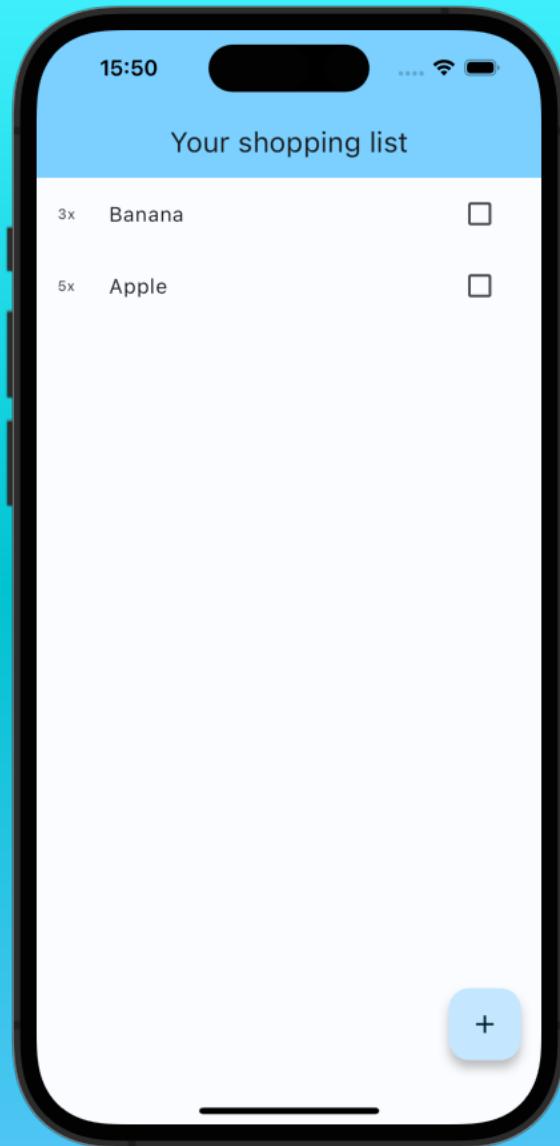
Checkbox 5



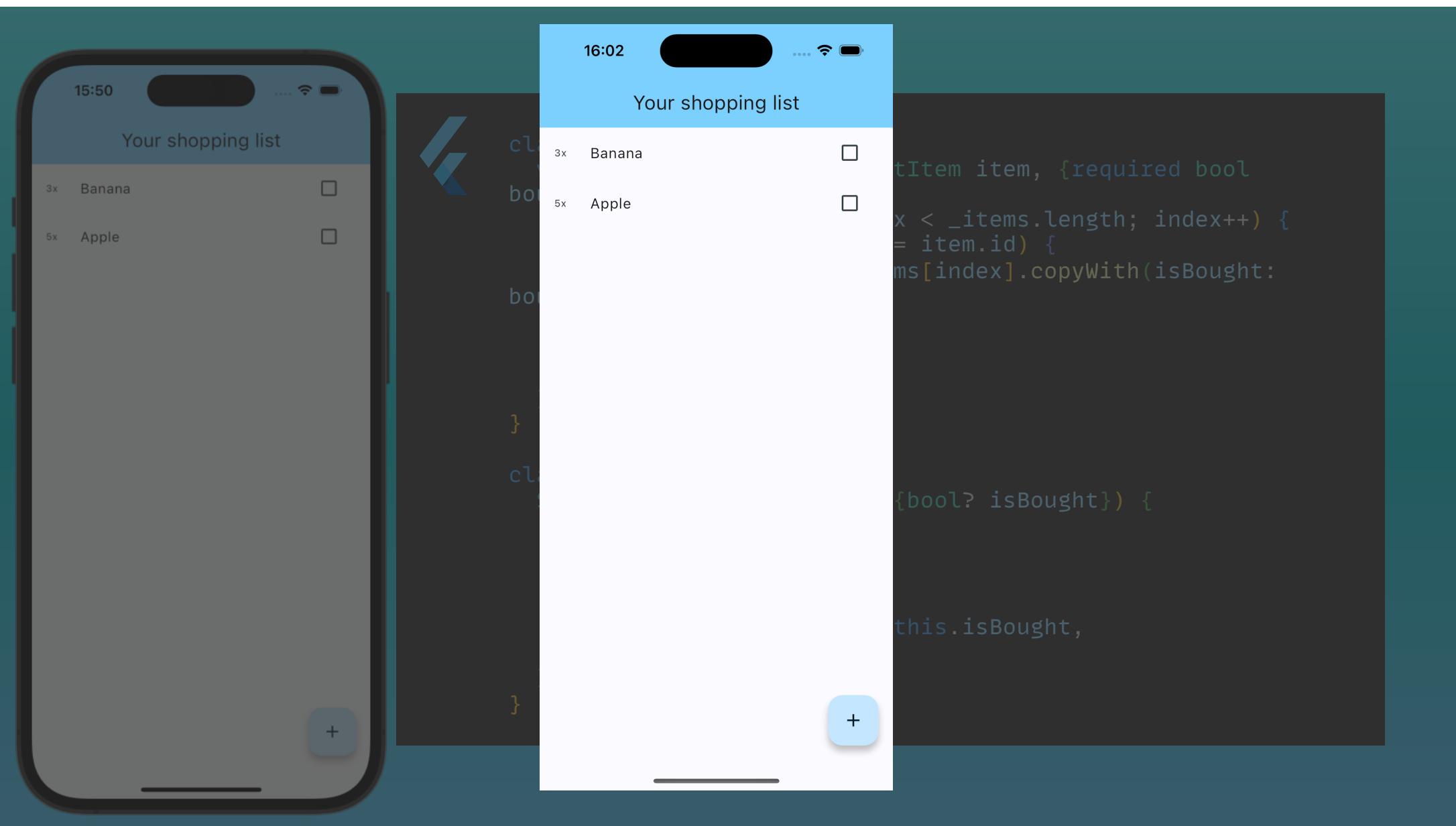
```
  @override  
  Widget build(BuildContext context) {  
    return ListView(  
      children: [  
        for (final item in _shoppingList.items)  
          ListTile(  
            leading: Text('${item.quantity}x'),  
            title: Text(item.name),  
            trailing: Checkbox(  
              value: false,  
              onChanged: (value) {},  
            ),  
          ),  
      ],  
    );  
  }
```



```
Flutter logo
@override
Widget build(BuildContext context) {
  return ListView(
    children: [
      for (final item in _shoppingList.items)
        ListTile(
          leading: Text('${item.quantity}x'),
          title: Text(item.name),
          trailing: Checkbox(
            value: item.isBought,
            onChanged: (value) {
              item.isBought = value!;
            },
          ),
        ),
    ],
}
```

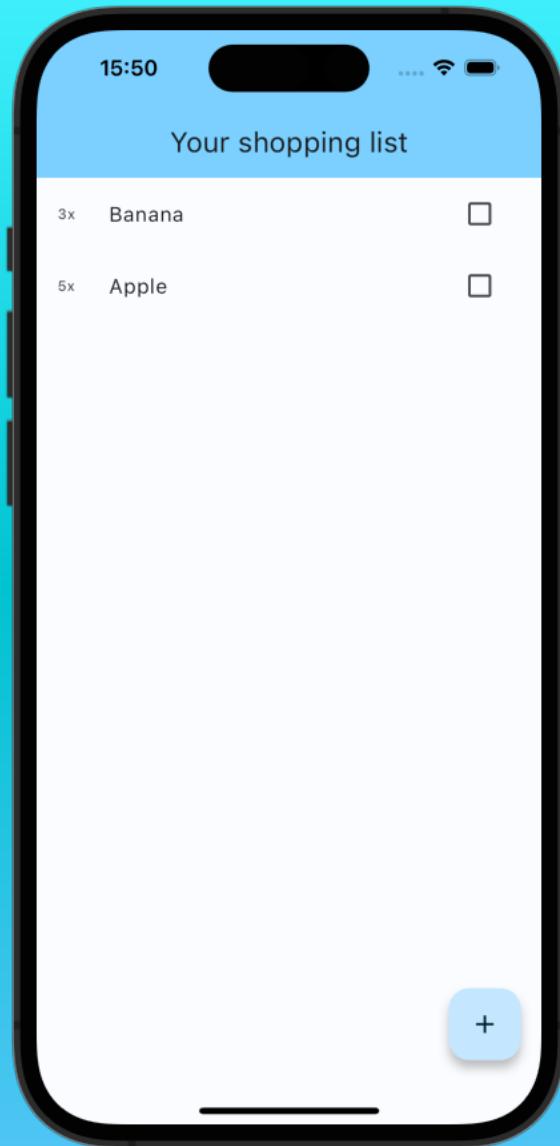


```
 @override  
Widget build(BuildContext context) {  
  return ListView(  
    children: [  
      for (final item in _shoppingList.items)  
        ListTile(  
          leading: Text('${item.quantity}x'),  
          title: Text(item.name),  
          trailing: Checkbox(  
            value: item.isBought,  
            onChanged: (value) {  
              item.isBought = value!;  
            },  
          ),  
        ),  
    ],  
  );  
}  
  
Null assertion operator ↗
```

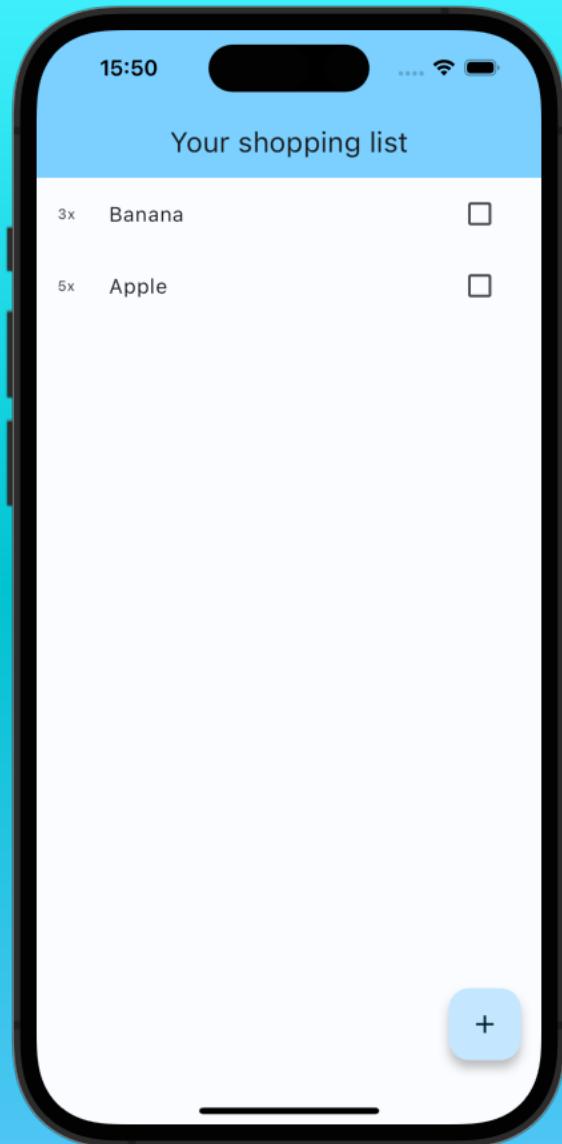


Simple State Management

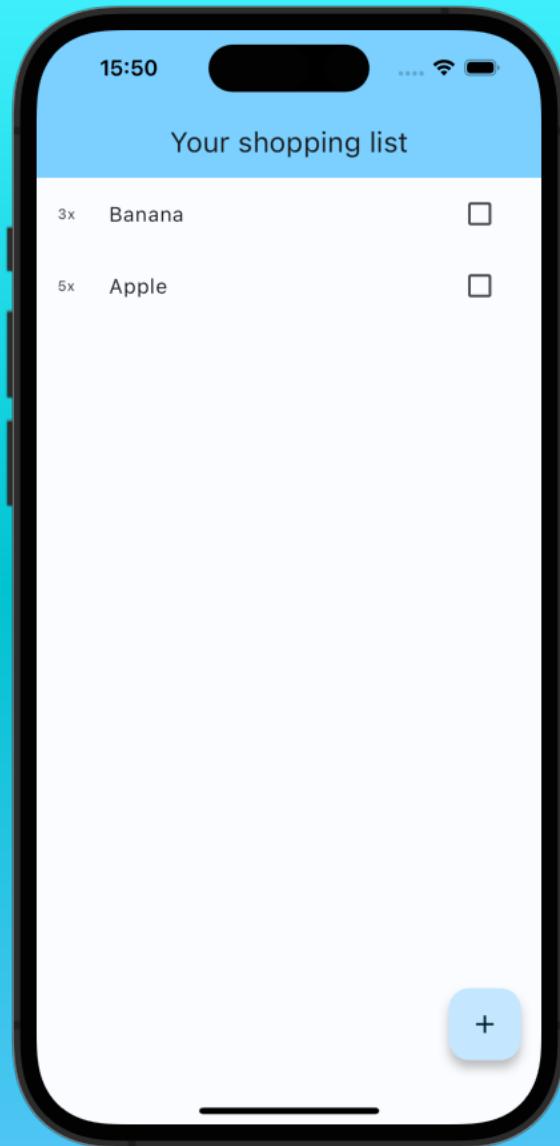
Shopping List App



```
Flutter logo
@override
Widget build(BuildContext context) {
  return ListView(
    children: [
      for (final item in _shoppingList.items)
        ListTile(
          leading: Text('${item.quantity}x'),
          title: Text(item.name),
          trailing: Checkbox(
            value: item.isBought,
            onChanged: (value) {
              item.isBought = value!;
            },
          ),
        ),
    ],
}
```

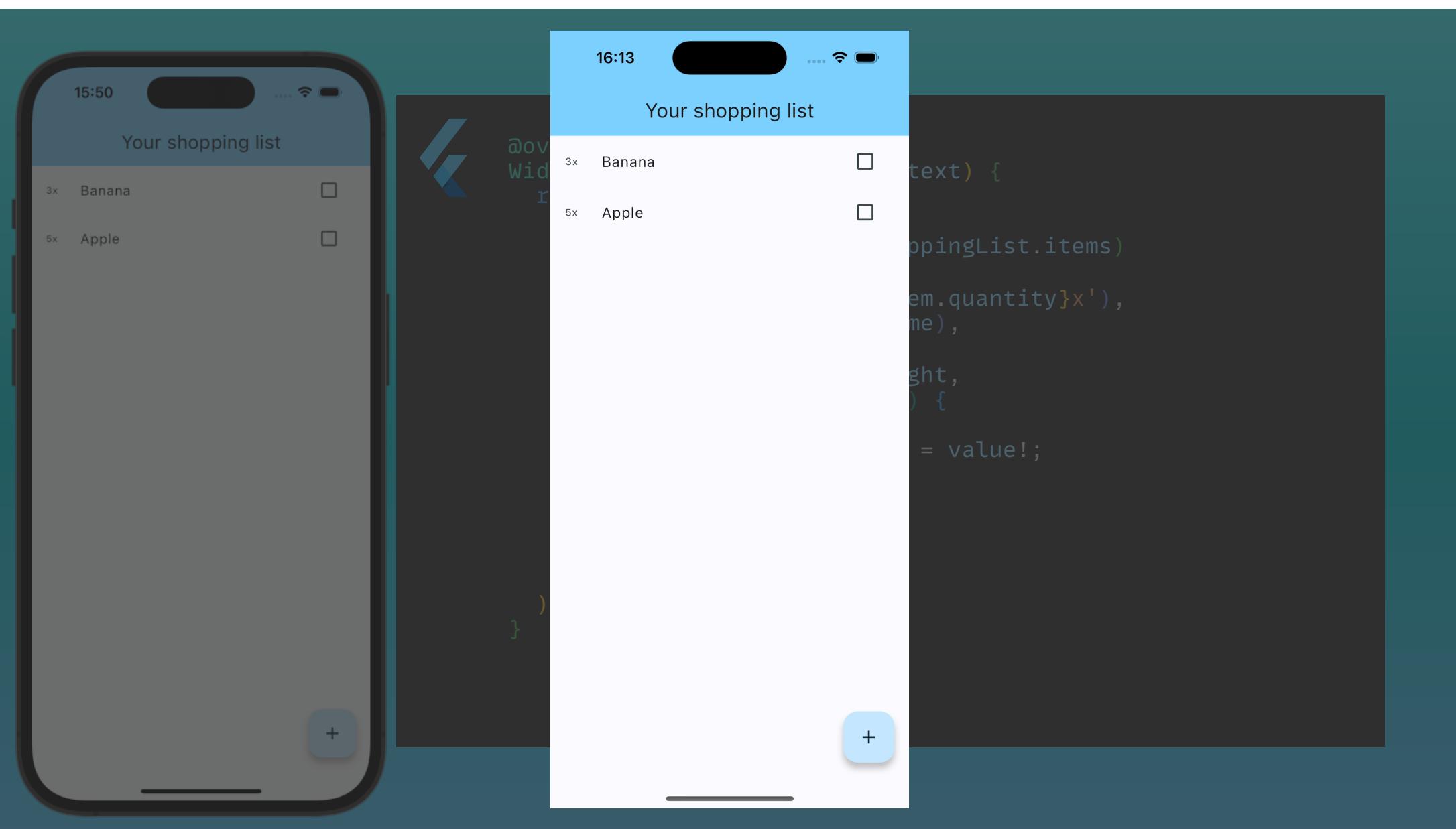


```
 @override  
Widget build(BuildContext context) {  
  return ListView(  
    children: [  
      for (final item in _shoppingList.items)  
        ListTile(  
          leading: Text('${item.quantity}x'),  
          title: Text(item.name),  
          trailing: Checkbox(  
            value: item.isBought,  
            onChanged: (value) {  
              item.isBought = value!;  
            },  
          ),  
        ),  
    ],  
  );  
}  
  
👉 We have to tell Flutter that the state changed
```



```
 @override  
Widget build(BuildContext context) {  
  return ListView(  
    children: [  
      for (final item in _shoppingList.items)  
        ListTile(  
          leading: Text('${item.quantity}x'),  
          title: Text(item.name),  
          trailing: Checkbox(  
            value: item.isBought,  
            onChanged: (value) {  
              setState(() {  
                item.isBought = value!;  
              });  
            },  
          ),  
        ),  
    ],  
  );  
}
```

👉 setState rerenders the whole widget sub tree



15:50

Your shopping list

3x Banana



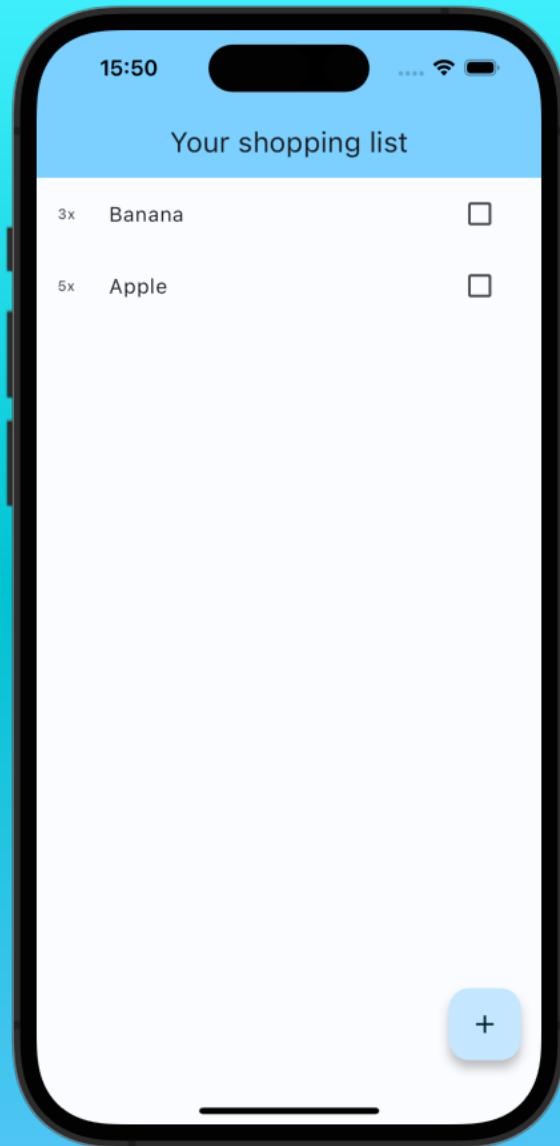
5x Apple



```
@override  
Widget build(BuildContext context) {  
  return ListView(  
    children: [  
      for (final item in _shoppingList.items)  
        ListTile(  
          leading: Text('${item.quantity}x'),  
          title: Text(item.name),  
          trailing: Checkbox(  
            value: item.isBought,  
            onChanged: (value) {  
              setState(() {  
                item.isBought = value!;  
              });  
            },  
          ),  
        ),  
    ],  
  );  
}
```

Toggling the Checkbox with ListTile click

ShoppingListApp



```
  @override  
  Widget build(BuildContext context) {  
    return ListView(  
      children: [  
        for (final item in _shoppingList.items)  
          ListTile(  
            leading: Text('${item.quantity}x'),  
            title: Text(item.name),  
            trailing: Checkbox(  
              value: item.isBought,  
              onChanged: (value) {  
                setState(() {  
                  item.isBought = value!;  
                });  
              },  
            ),  
          ),  
      ],  
    );  
  }  
}
```

15:50

Your shopping list

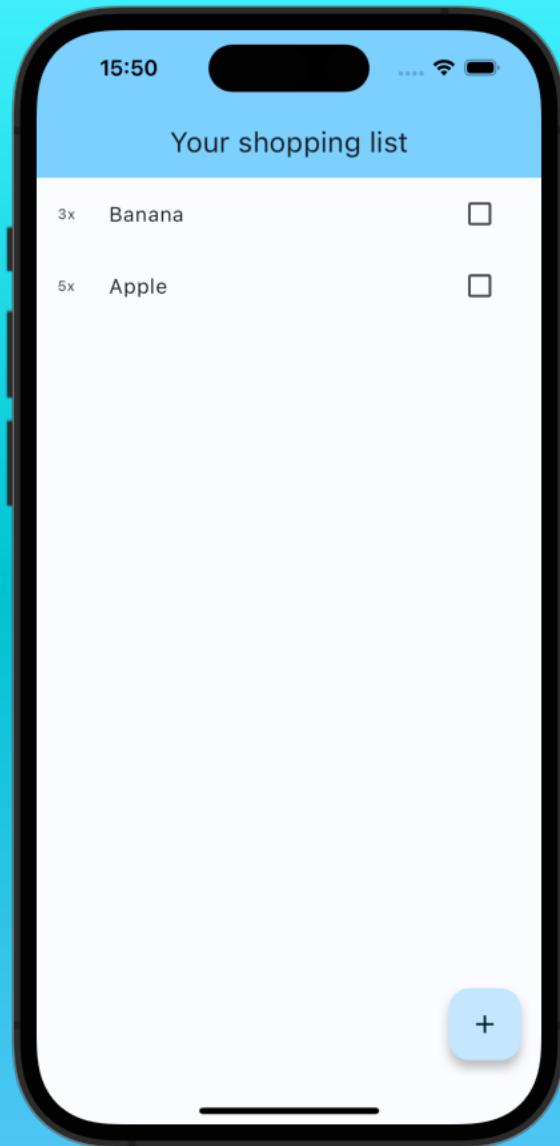
3x Banana



5x Apple

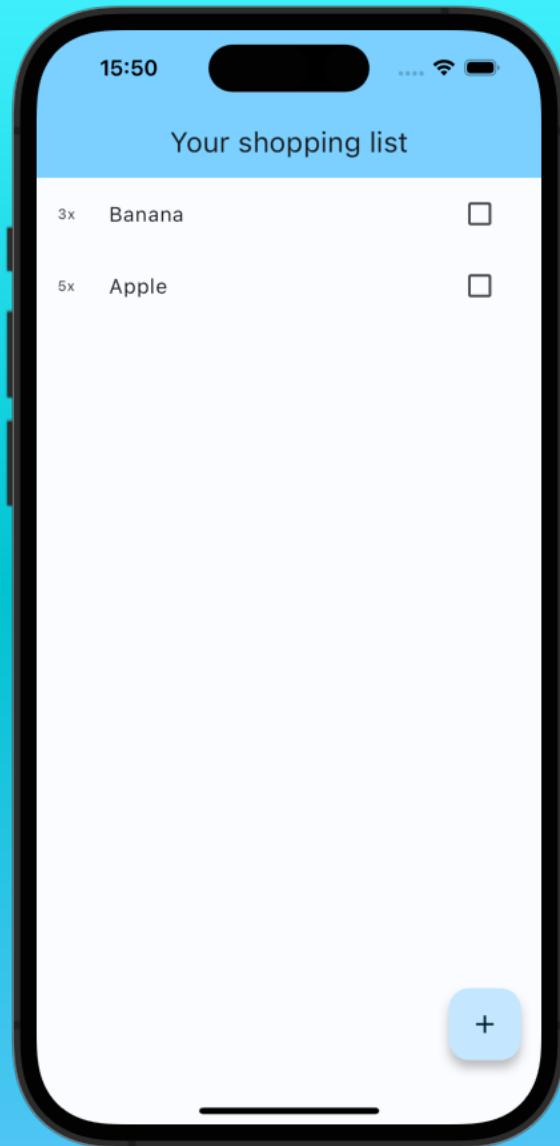


```
ListTile(  
    onTap: () {  
        setState(() {  
            item.isBought = !item.isBought;  
        });  
    },  
    leading: Text('${item.quantity}x'),  
    title: Text(item.name),  
    trailing: Checkbox(  
        value: item.isBought,  
        onChanged: (value) {  
            setState(() {  
                item.isBought = value!;  
            });  
        },  
    ),  
)
```



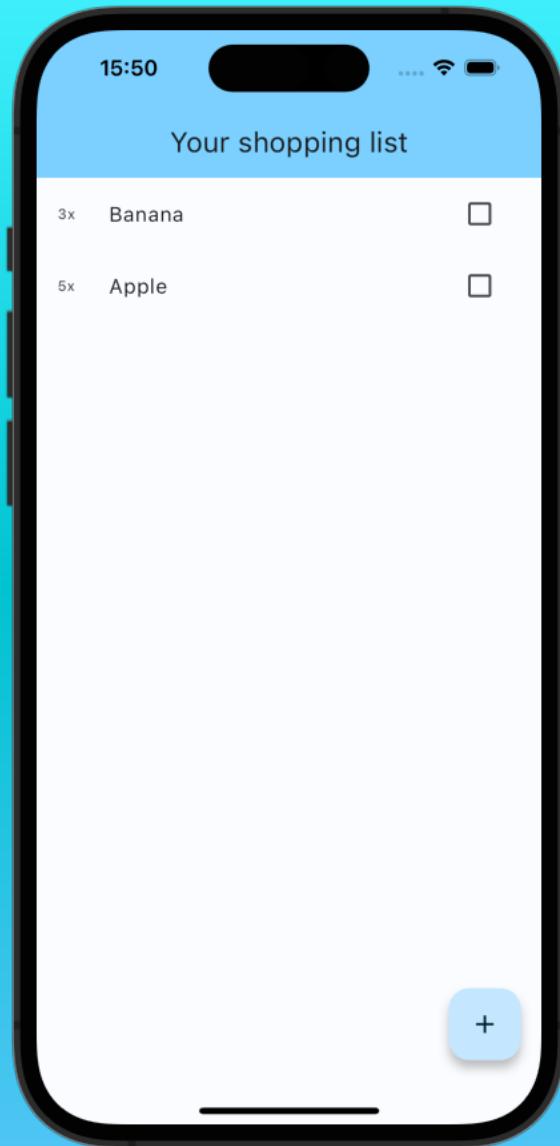
```
 ListTile(  
    onTap: () {  
        setState(() {  
            item.isBought = !item.isBought;  
        });  
    },  
    leading: Text('${item.quantity}x'),  
    title: Text(item.name),  
    trailing: Checkbox(  
        value: item.isBought,  
        onChanged: (value) {  
            setState(() {  
                item.isBought = value!;  
            });  
        },  
    ),  
)
```

Duplicated code

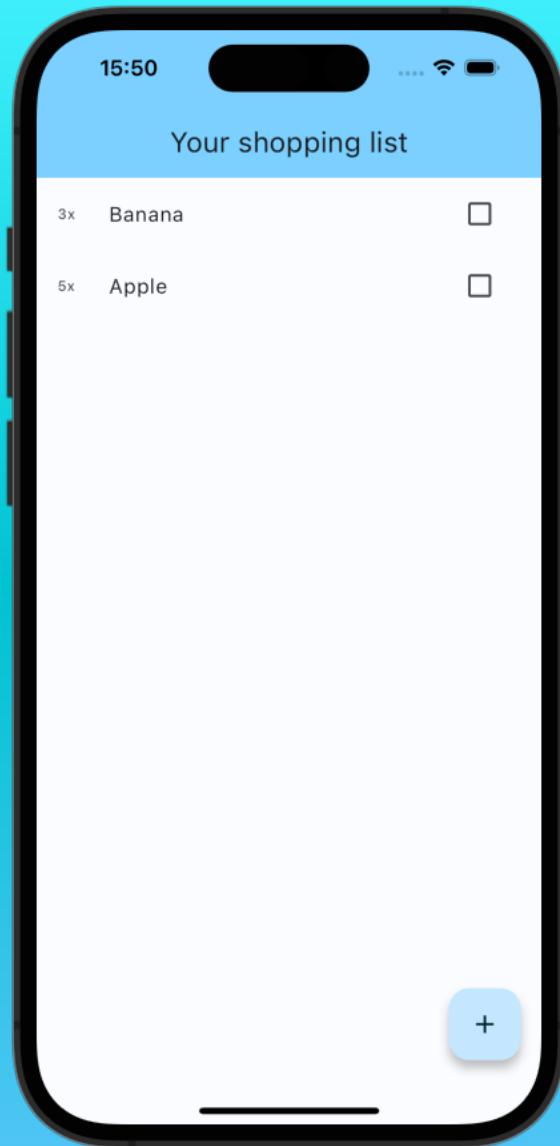


```
void _toggleBought(ShoppingListIteм item) {  
    setState(() {  
        item.isBought = !item.isBought;  
    });  
}
```

A dark grey rectangular area contains a snippet of Kotlin code. The code defines a function named "_toggleBought" that takes a parameter of type "ShoppingListIteм". Inside the function, it calls "setState" with a lambda expression. The lambda expression toggles the value of "item.isBought". The code ends with a closing brace for the function. To the left of the code, the blue Flutter logo is displayed.



```
 ListTile(  
    onTap: () {  
        setState(() {  
            item.isBought = !item.isBought;  
        });  
    },  
    leading: Text('${item.quantity}x'),  
    title: Text(item.name),  
    trailing: Checkbox(  
        value: item.isBought,  
        onChanged: (value) {  
            setState(() {  
                item.isBought = value!;  
            });  
        },  
    ),  
)
```



```
 ListTile(  
    onTap: () {  
        _toggleBought(item);  
    },  
    leading: Text('${item.quantity}x'),  
    title: Text(item.name),  
    trailing: Checkbox(  
        value: item.isBought,  
        onChanged: (_) {  
            _toggleBought(item);  
        },  
    ),  
)
```

Reflecting

Shopping List App

15:50

Your shopping list

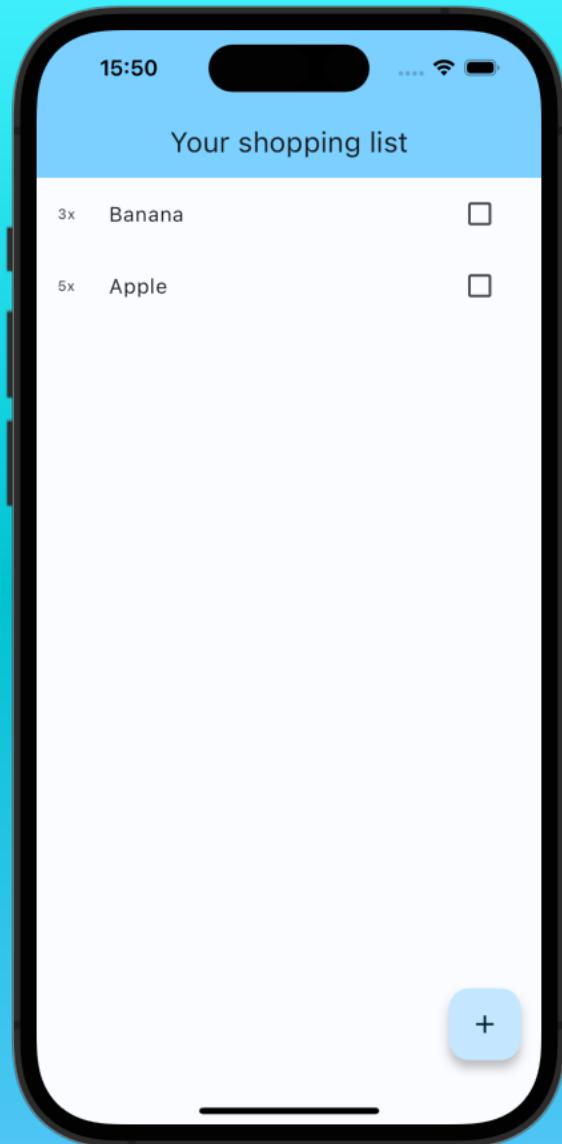
3x Banana



5x Apple



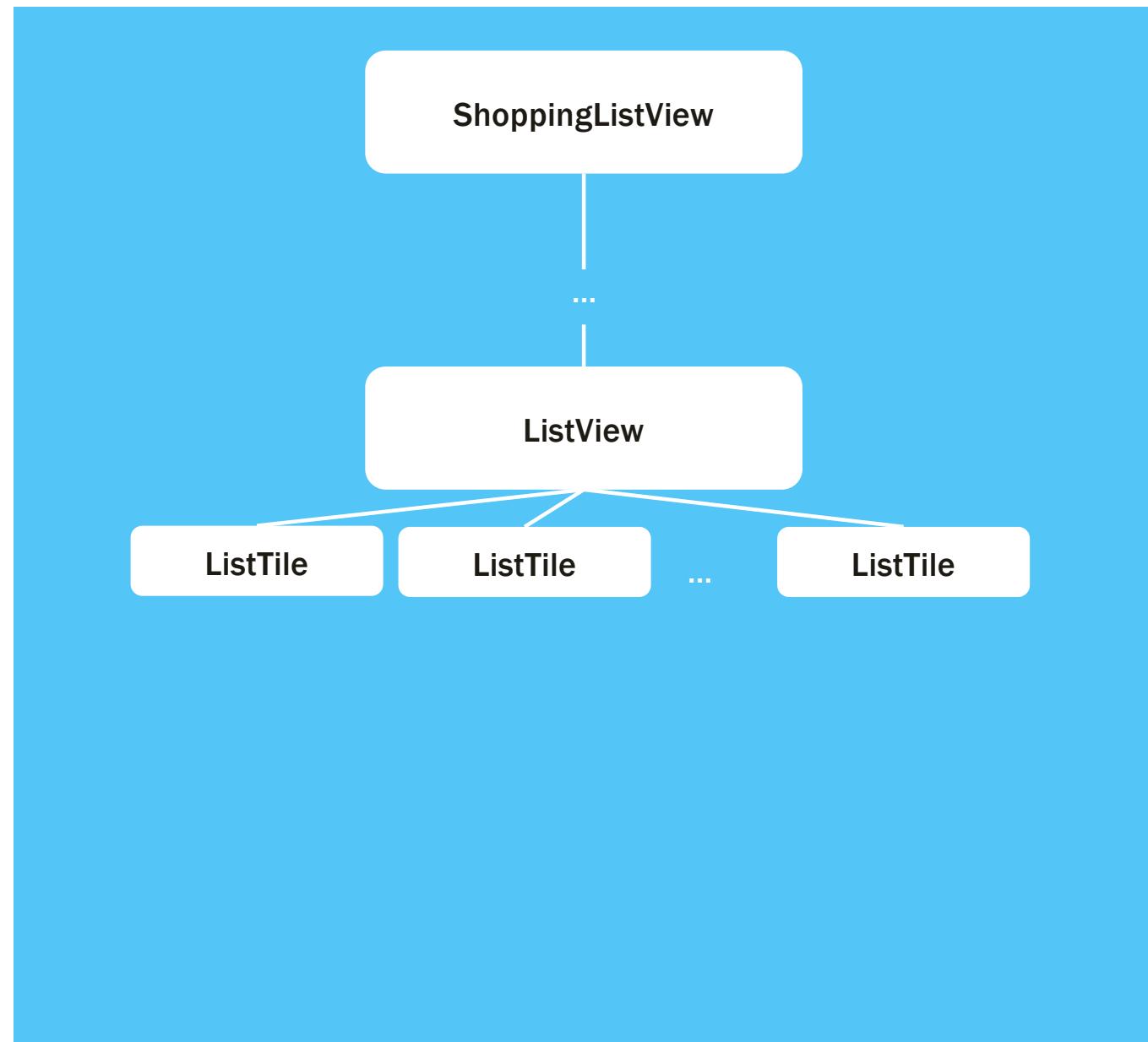
```
@override  
Widget build(BuildContext context) {  
  return ListView(  
    children: [  
      for (final item in _shoppingList.items)  
        ListTile(  
          onTap: () {  
            _toggleBought(item);  
          },  
          leading: Text('${item.quantity}x'),  
          title: Text(item.name),  
          trailing: Checkbox(  
            value: item.isBought,  
            onChanged: (_) {  
              _toggleBought(item);  
            },  
          ),  
        ),  
    ],  
  );  
}
```

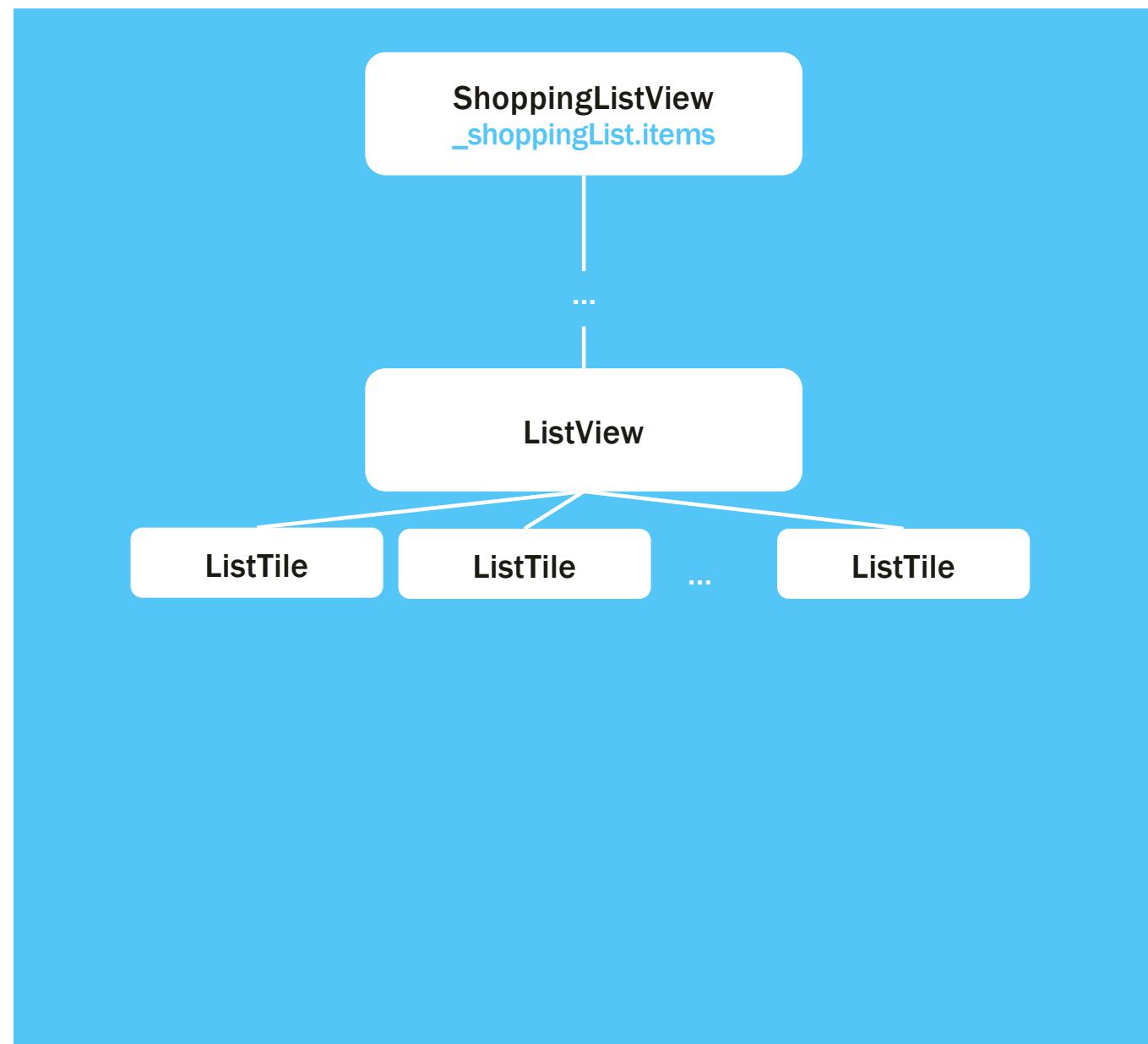
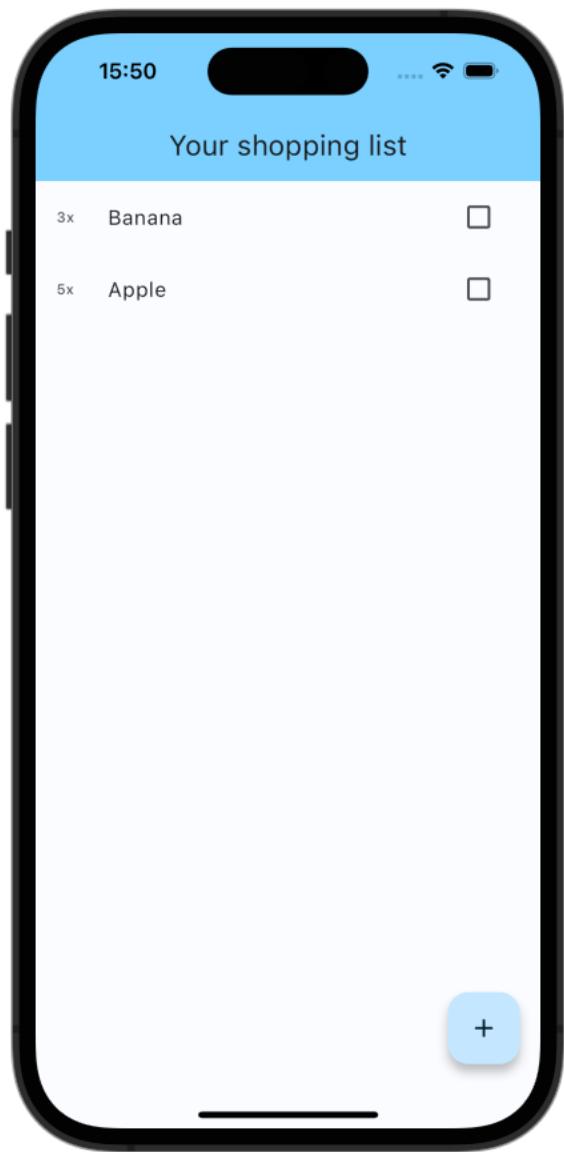


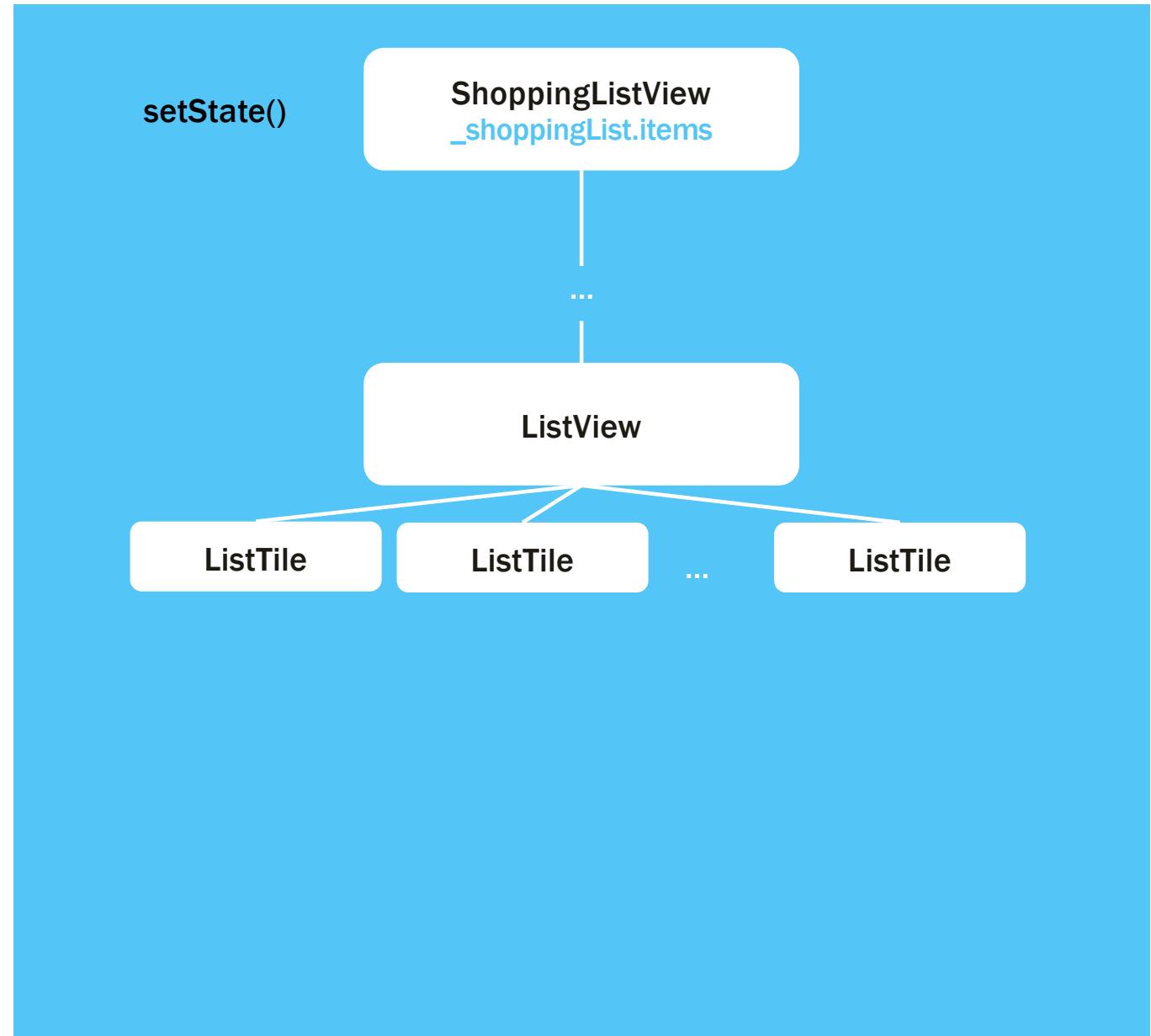
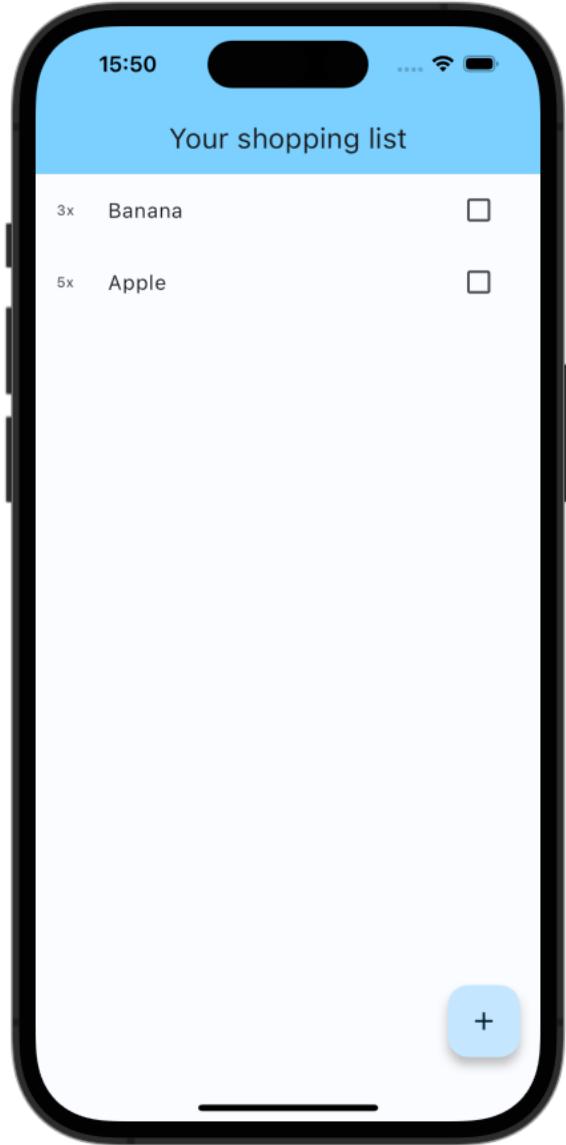
```
 @override  
Widget build(BuildContext context) {  
  return ListView(  
    children: [  
      for (final item in _shoppingList.items)  
        ListTile(  
          onTap: () {  
            _toggleBought(item);  
          },  
          leading: Text('${item.quantity}x'),  
          title: Text(item.name),  
          trailing: Checkbox(  
            value: item.isBought,  
            onChanged: (_) {  
              _toggleBought(item);  
            }  
          ),  
        ),  
    ],  
  );  
}  
  
👉 rerenders the whole widget sub tree
```

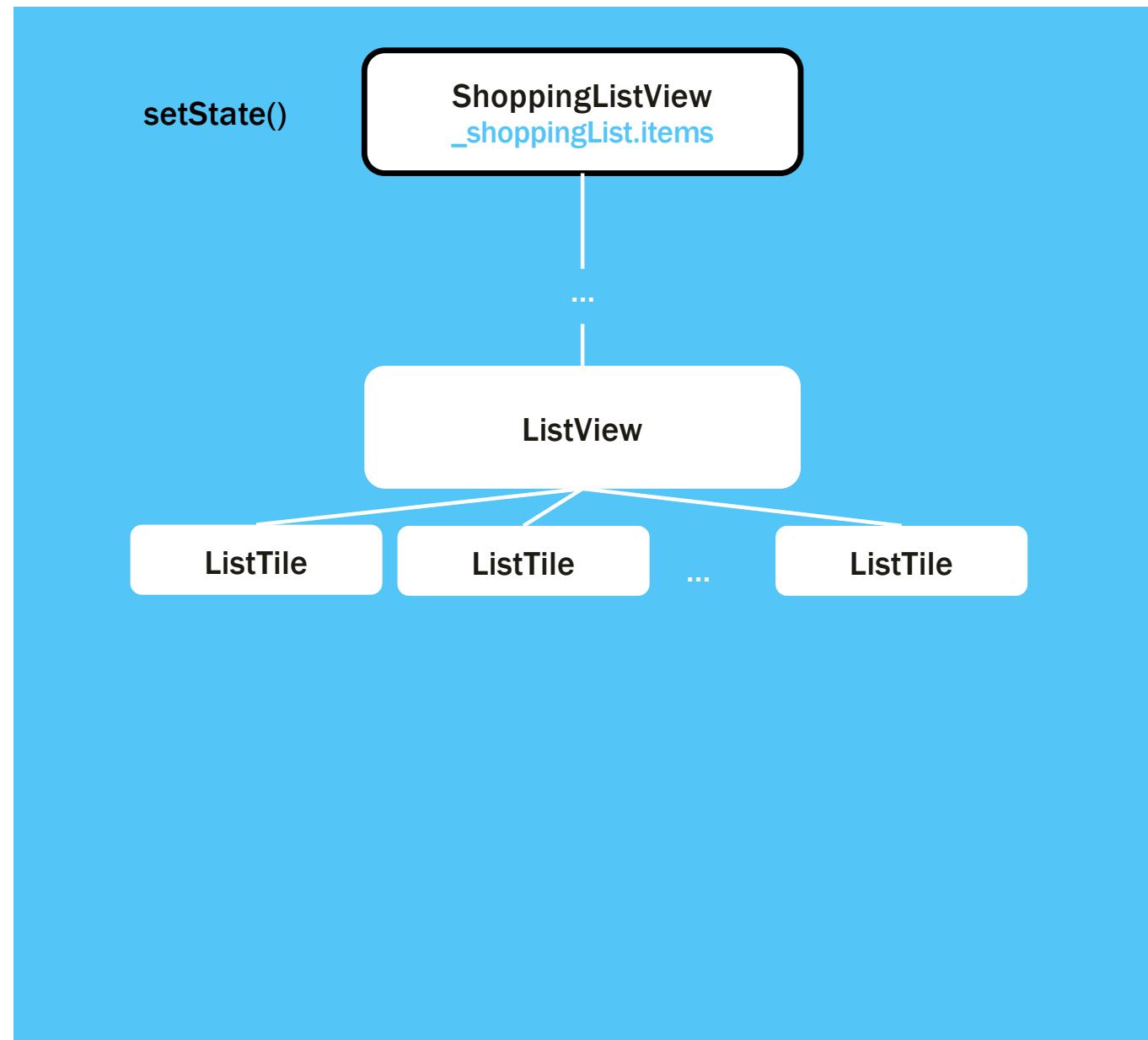
Why does setState rerender every ListTile?

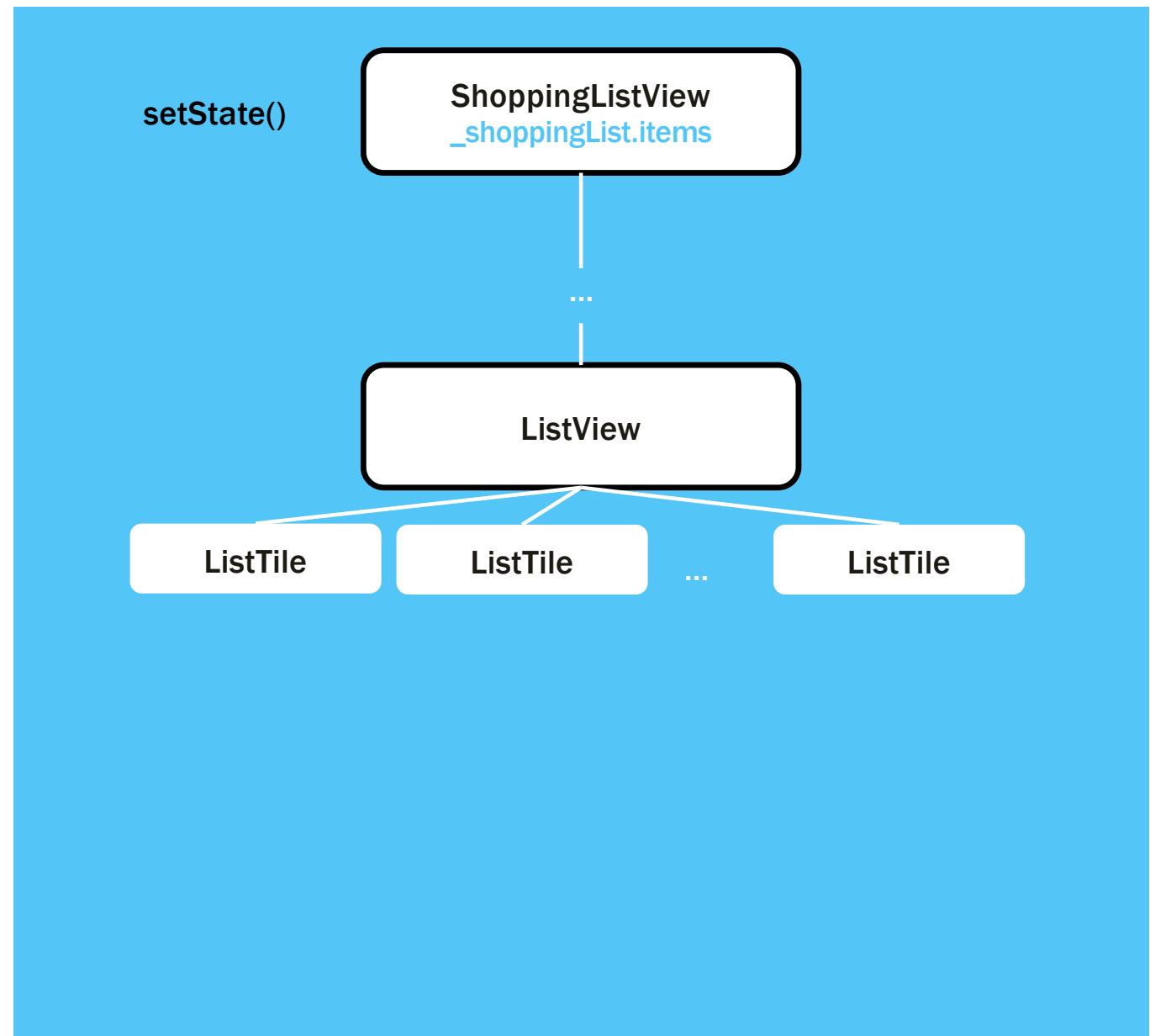
Shopping List App

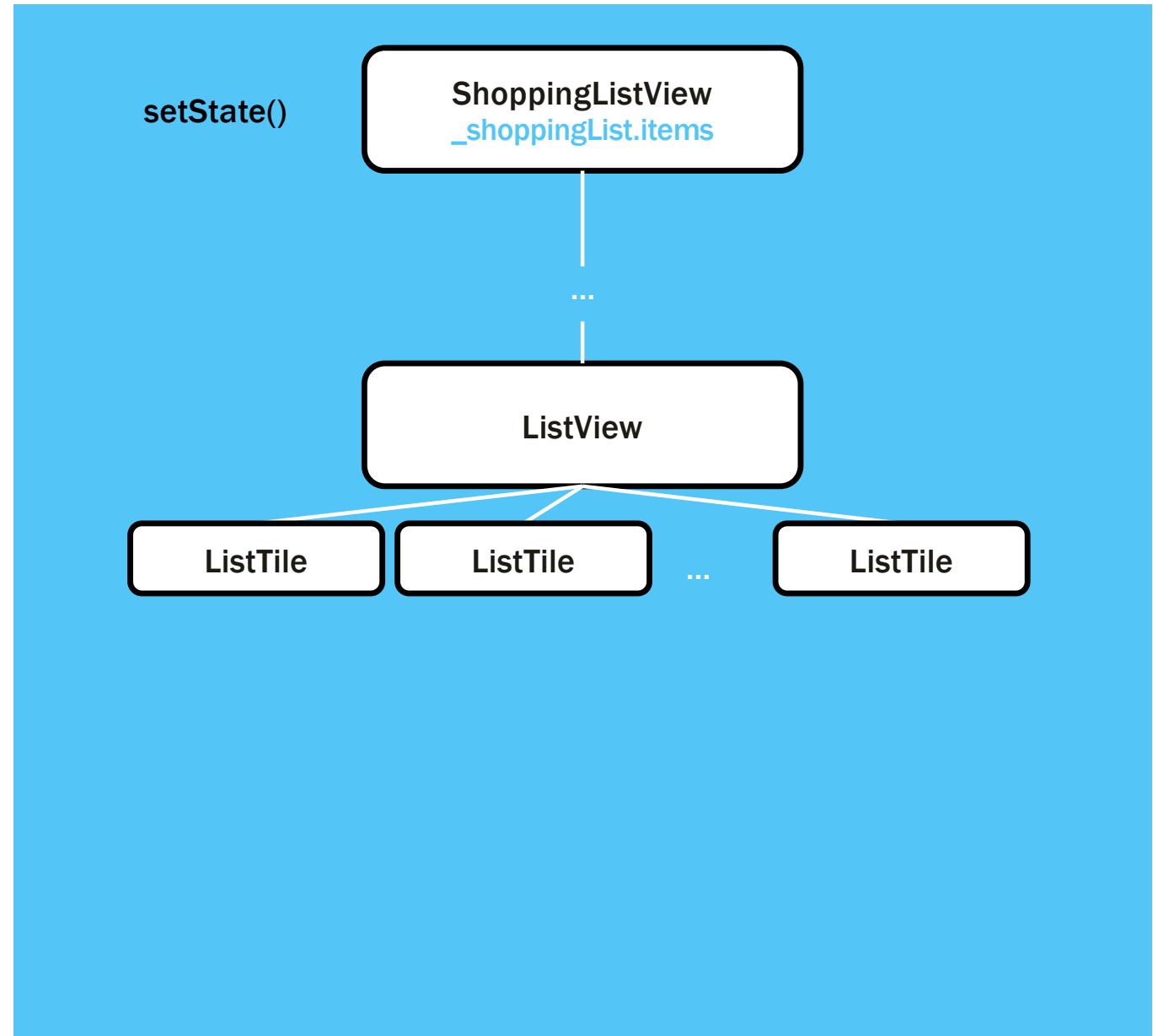
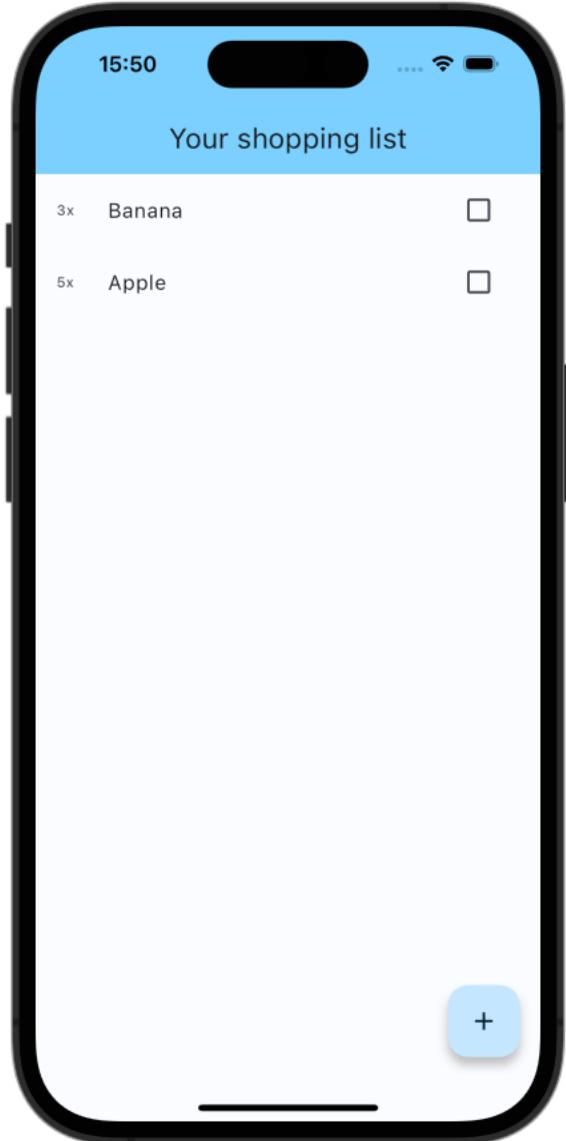






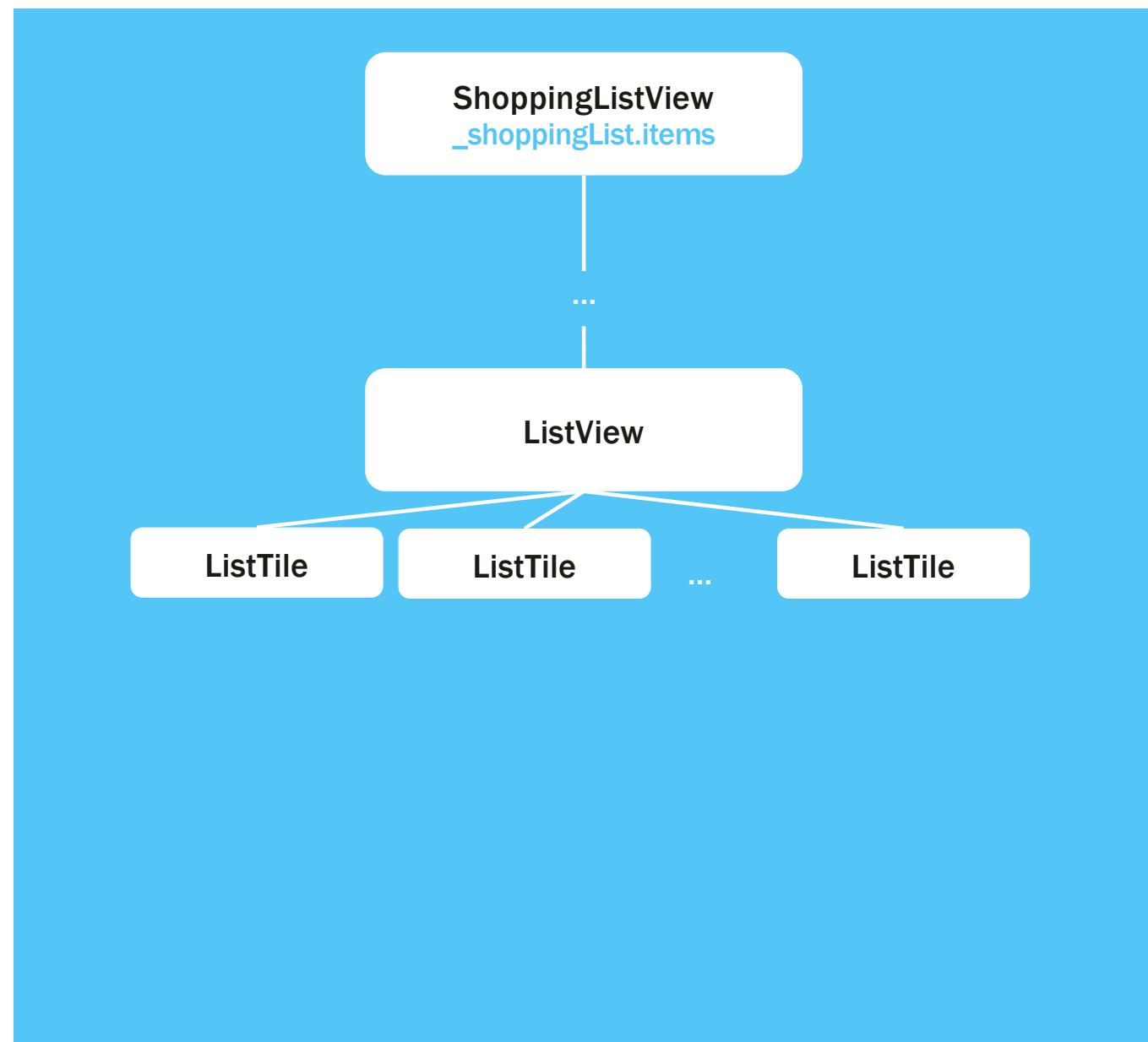
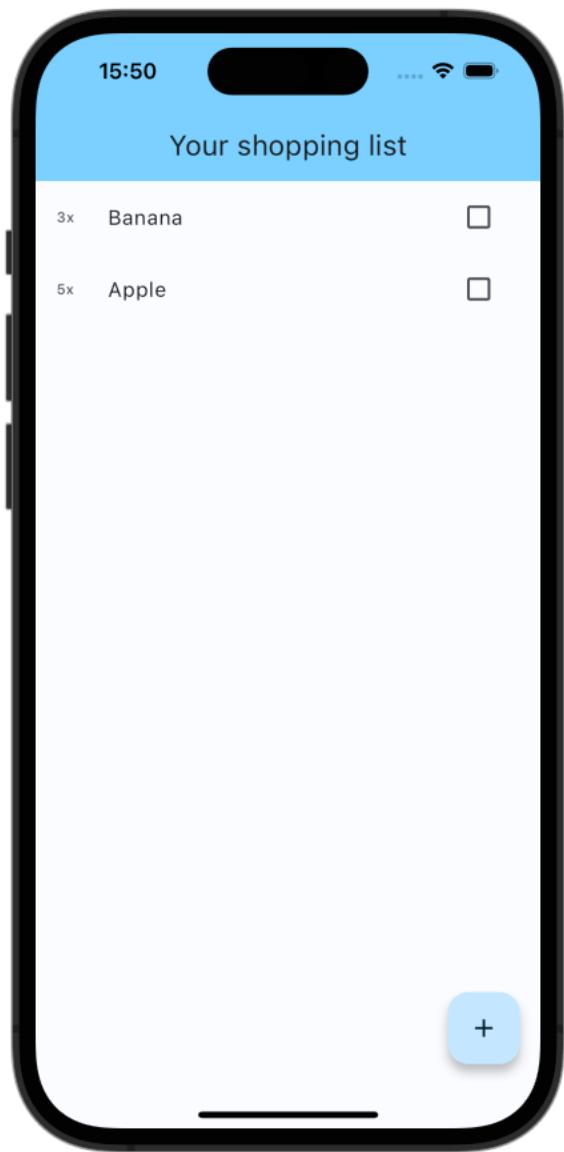


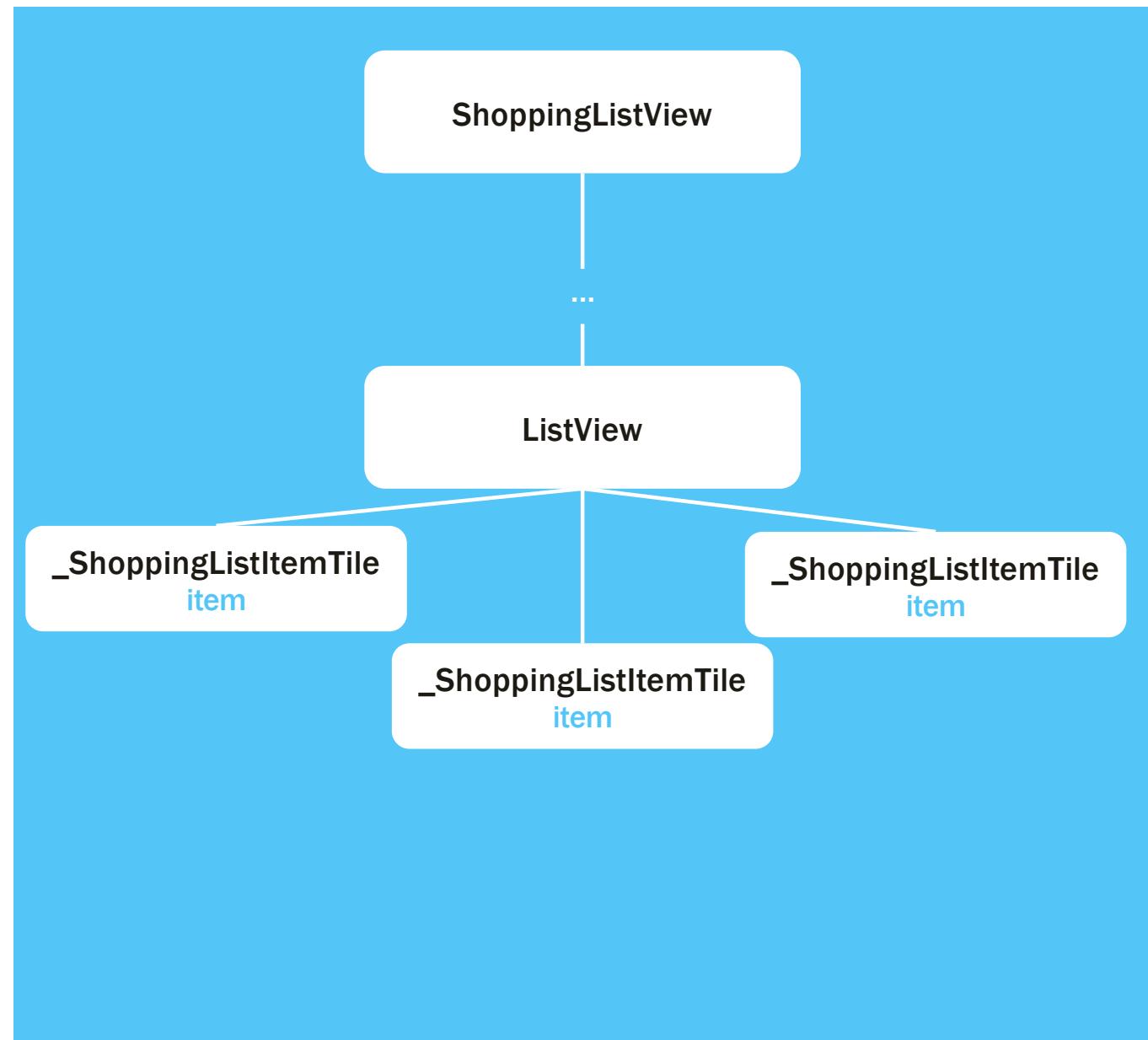


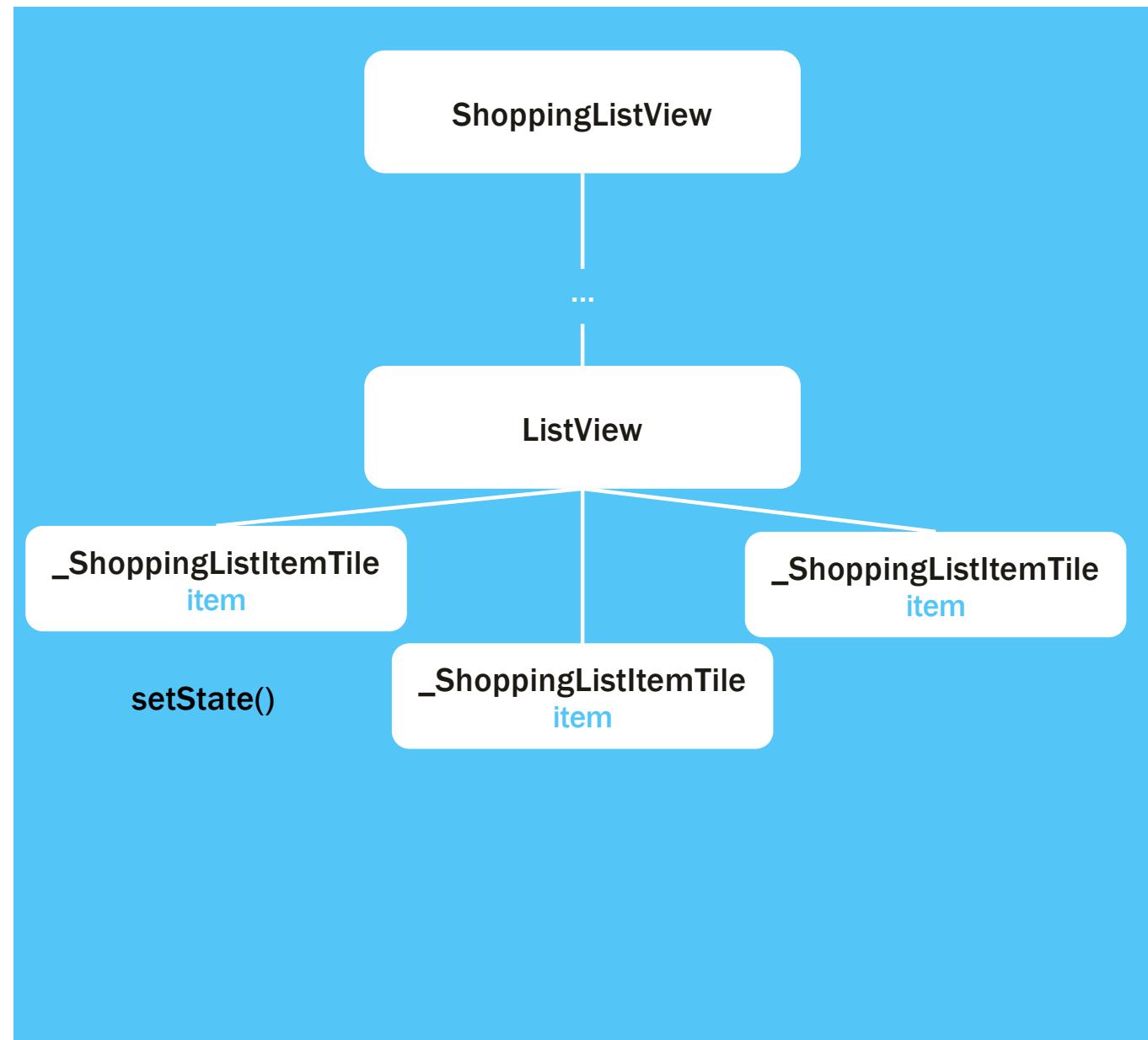


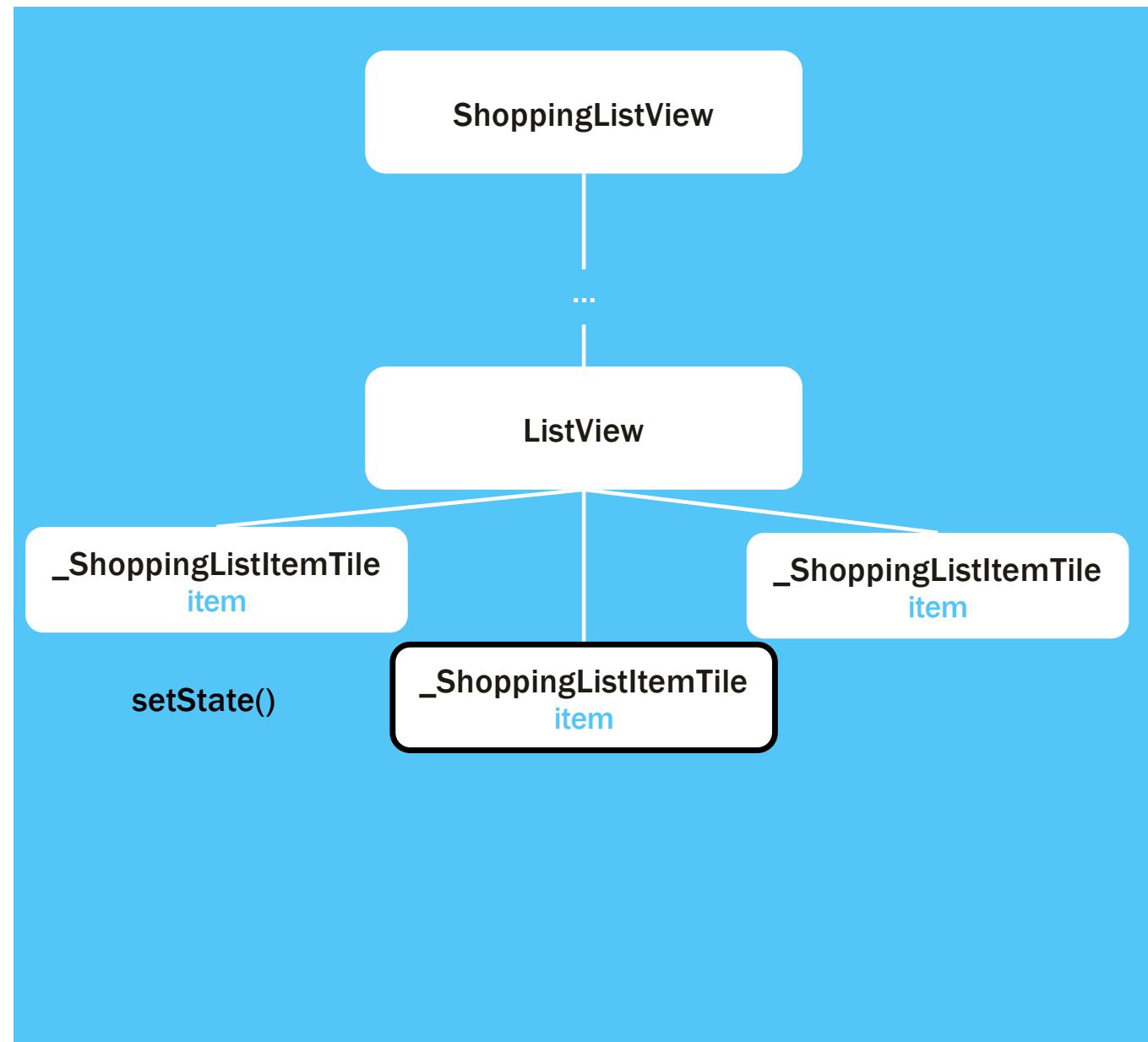
Lowering state

Shopping List App



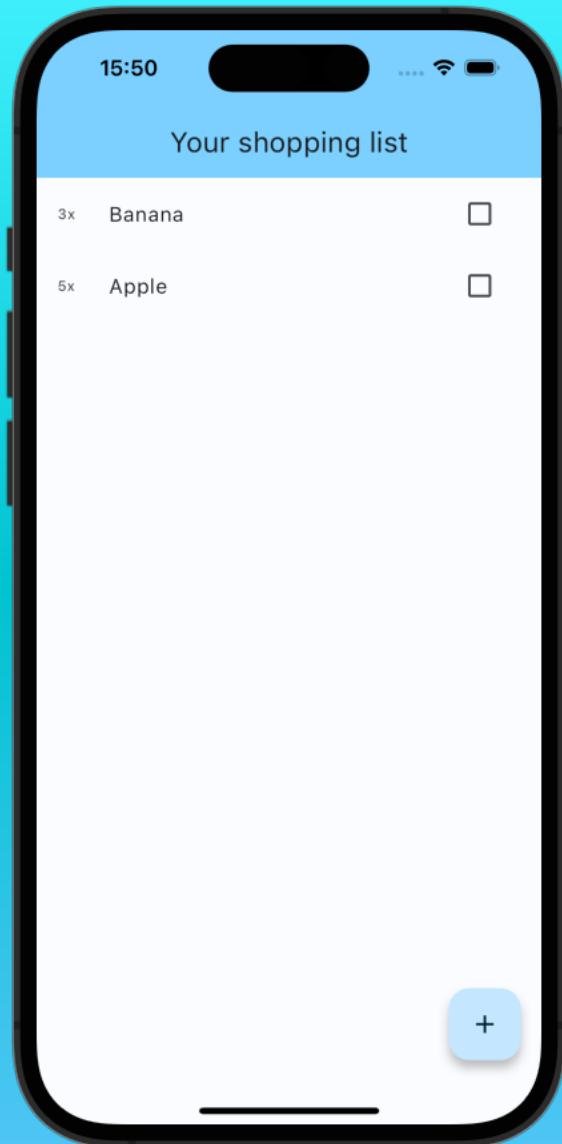






Implementation

Shopping List App

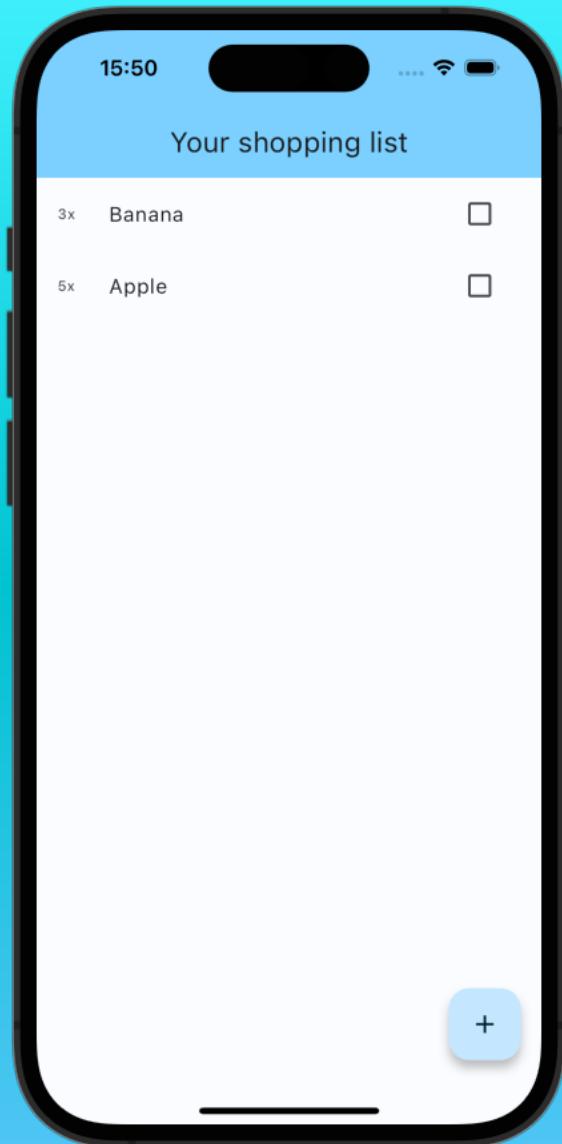


```
class _ShoppingListTile extends StatefulWidget {
  const _ShoppingListTile({
    required this.item,
    super.key,
  });

  final ShoppingListItem item;

  @override
  State<_ShoppingListTile> createState() =>
  _ShoppingListTileState();
}

class _ShoppingListTileState extends State<_ShoppingListTile> {
  @override
  Widget build(BuildContext context) {
    return const Placeholder();
}
```



```
class _ShoppingListTileState extends State<_ShoppingListTile> {
  @override
  Widget build(BuildContext context) {
    return ListTile(
      onTap: () {
        _toggleBought(item);
      },
      leading: Text('${item.quantity}x'),
      title: Text(item.name),
      trailing: Checkbox(
        value: item.isBought,
        onChanged: (value) {
          _toggleBought(item);
        },
      );
    }
}
```

15:50

Your shopping list

3x Banana

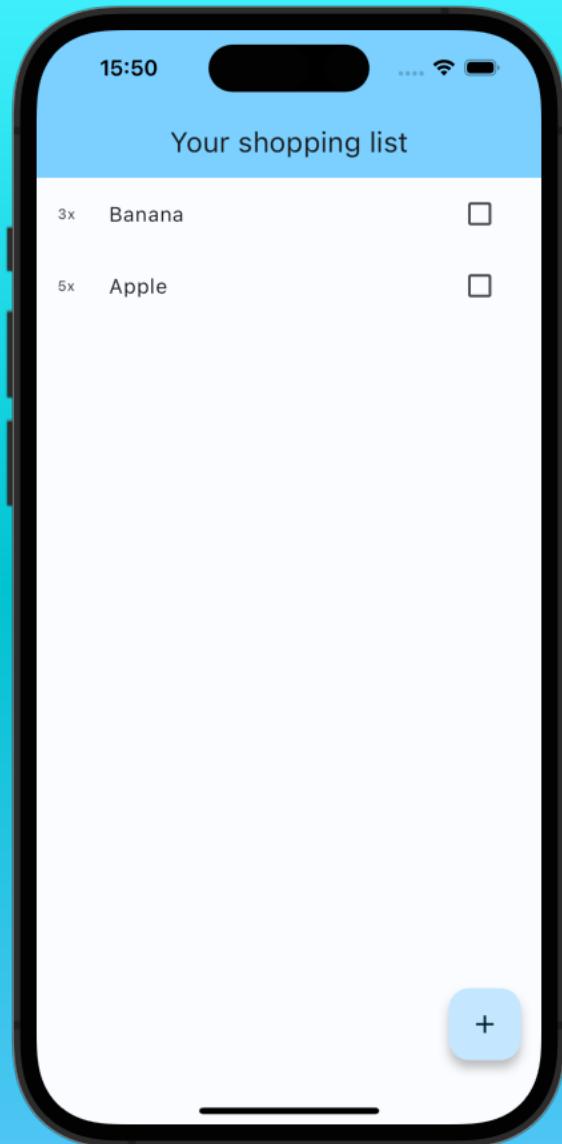


5x Apple



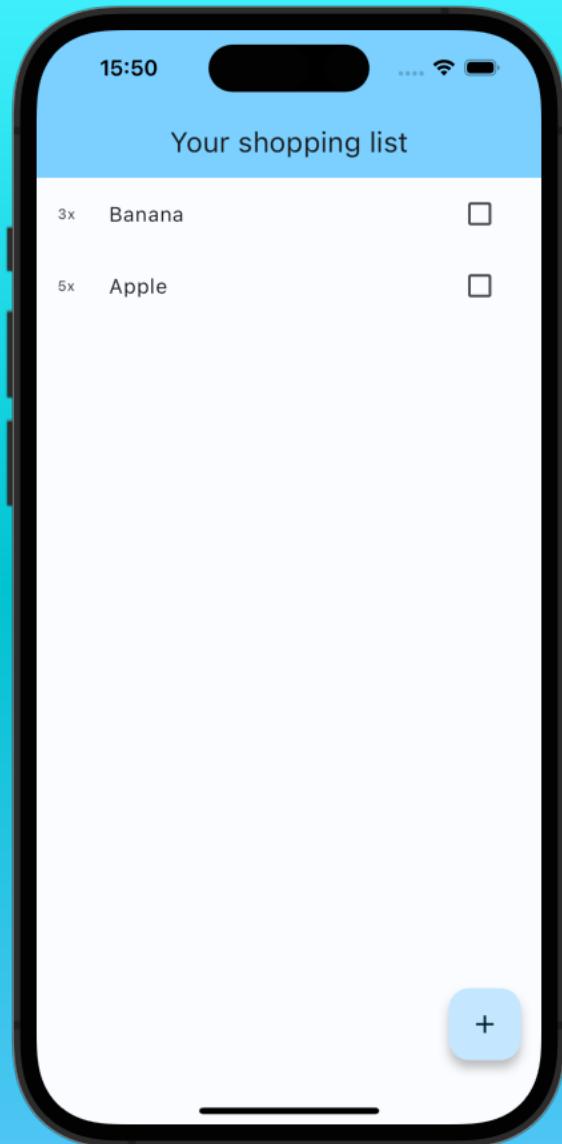
```
class _ShoppingListTileState extends State<_ShoppingListTile> {
  ShoppingListItem get item => widget.item;
  @override
  Widget build(BuildContext context) {
    return ListTile(
      onTap: () {
        _toggleBought(item);
      },
      leading: Text('${item.quantity}x'),
      title: Text(item.name),
      trailing: Checkbox(
        value: item.isBought,
        onChanged: (value) {
          _toggleBought(item);
        },
      ),
    );
  }
}
```

👉 access properties of StatefulWidget via widget.



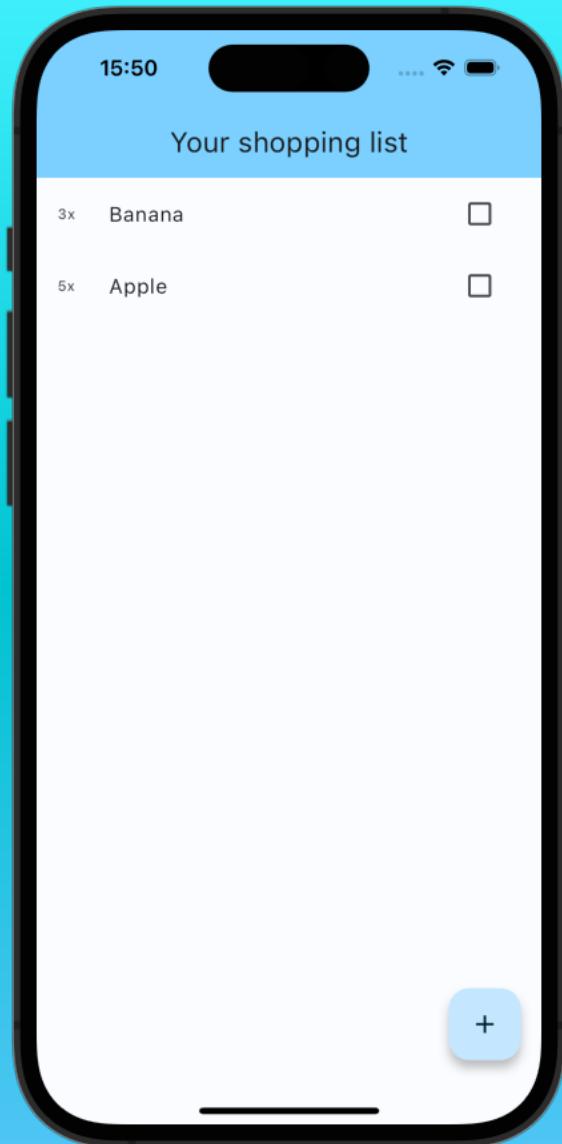
```
Flutter logo
ShoppingListItem get item => widget.item;
void _toggleBought() {
    setState(() {
        item.isBought = !item.isBought;
    });
}

@Override
Widget build(BuildContext context) {
    return ListTile(
        onTap: () {
            _toggleBought();
        },
        leading: Text('${item.quantity}x'),
        title: Text(item.name),
        trailing: Checkbox(
            value: item.isBought,
            onChanged: (value) {
                _toggleBought();
            },
        ),
    );
}
```



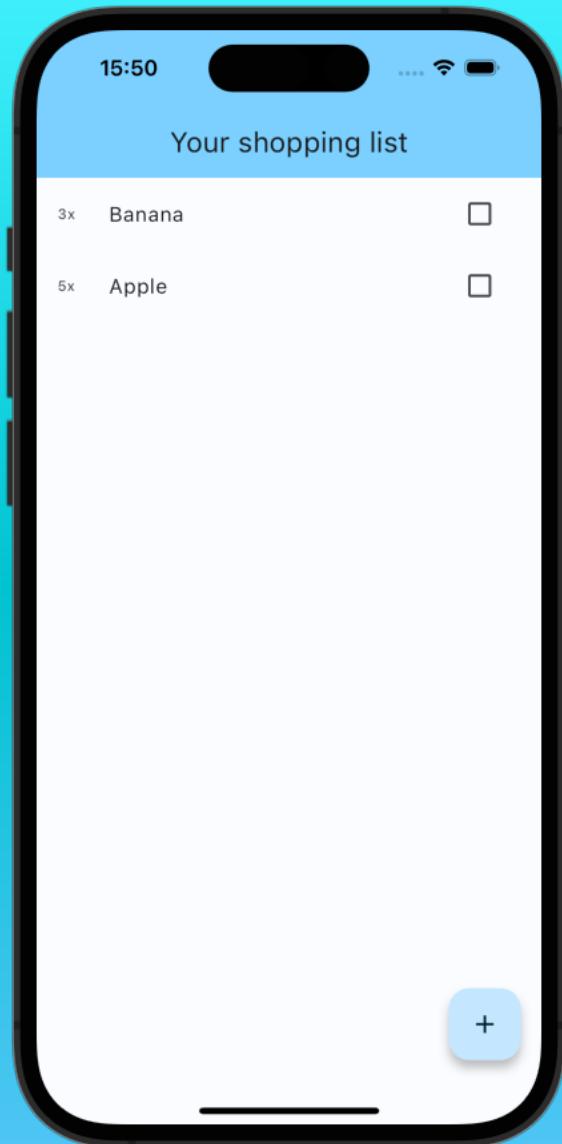
```
Flutter logo
ShoppingListItem get item => widget.item;
void _toggleBought() {
    setState(() {
        item.isBought = !item.isBought;
    });
}

@Override
Widget build(BuildContext context) {
    return ListTile(
        onTap: _toggleBought,
        leading: Text('${item.quantity}x'),
        title: Text(item.name),
        trailing: Checkbox(
            value: item.isBought,
            onChanged: (_)
            {
                _toggleBought();
            },
        ),
    );
}
```



```
void _toggleBought(ShoppingListTile item) {
    setState(() {
        item.isBought = !item.isBought;
    });
}

@Override
Widget build(BuildContext context) {
    return ListView(
        children: [
            for (final item in _shoppingList.items)
                _ShoppingListTile(
                    item: item,
                ),
        ],
    );
}
```



```
Flutter logo  
@override  
Widget build(BuildContext context) {  
  return ListView(  
    children: [  
      for (final item in _shoppingList.items)  
        _ShoppingListItemTile(  
          item: item,  
        ),  
    ],  
  );  
}
```

15:50

Your shopping list

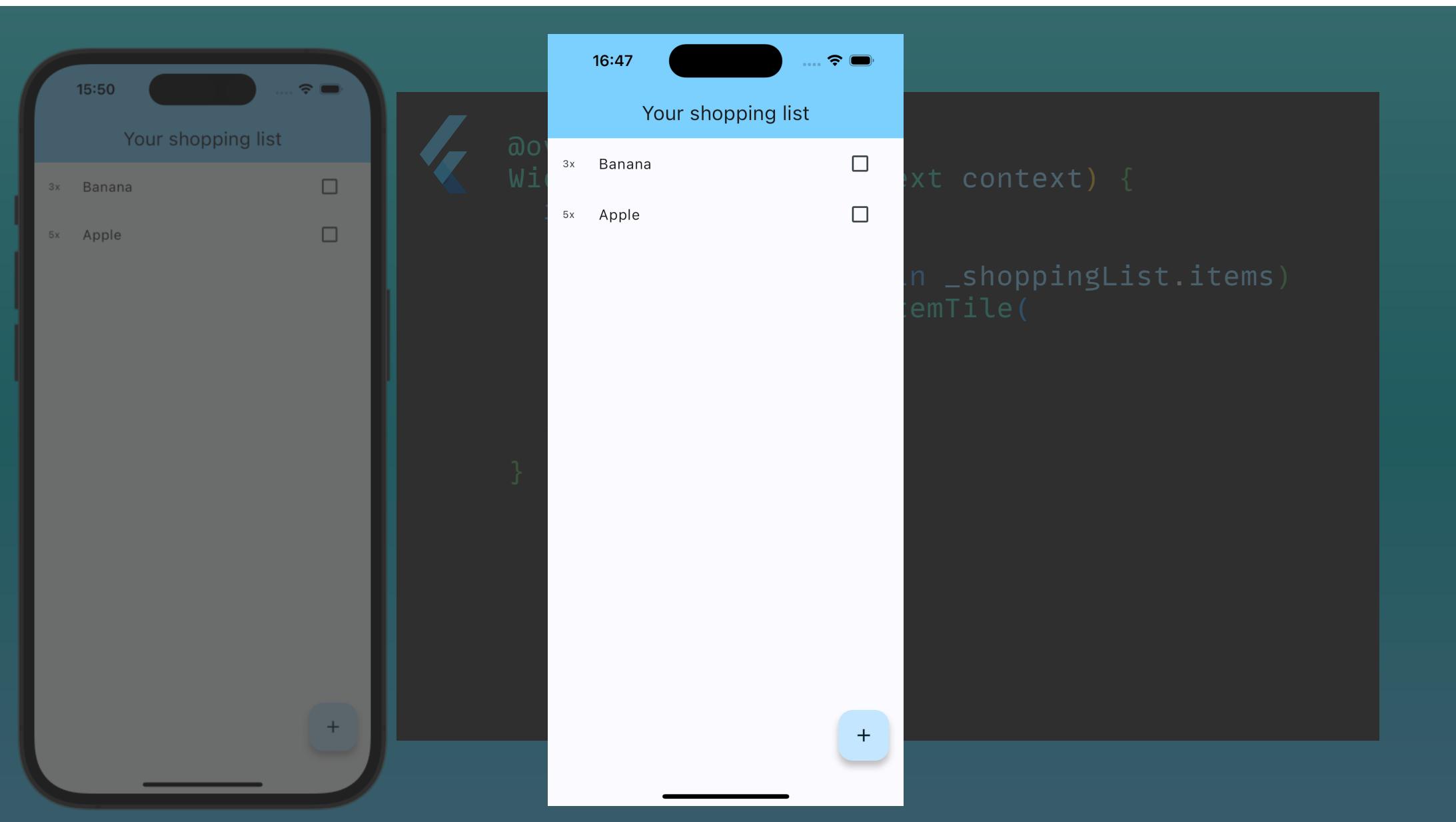
3x Banana



5x Apple

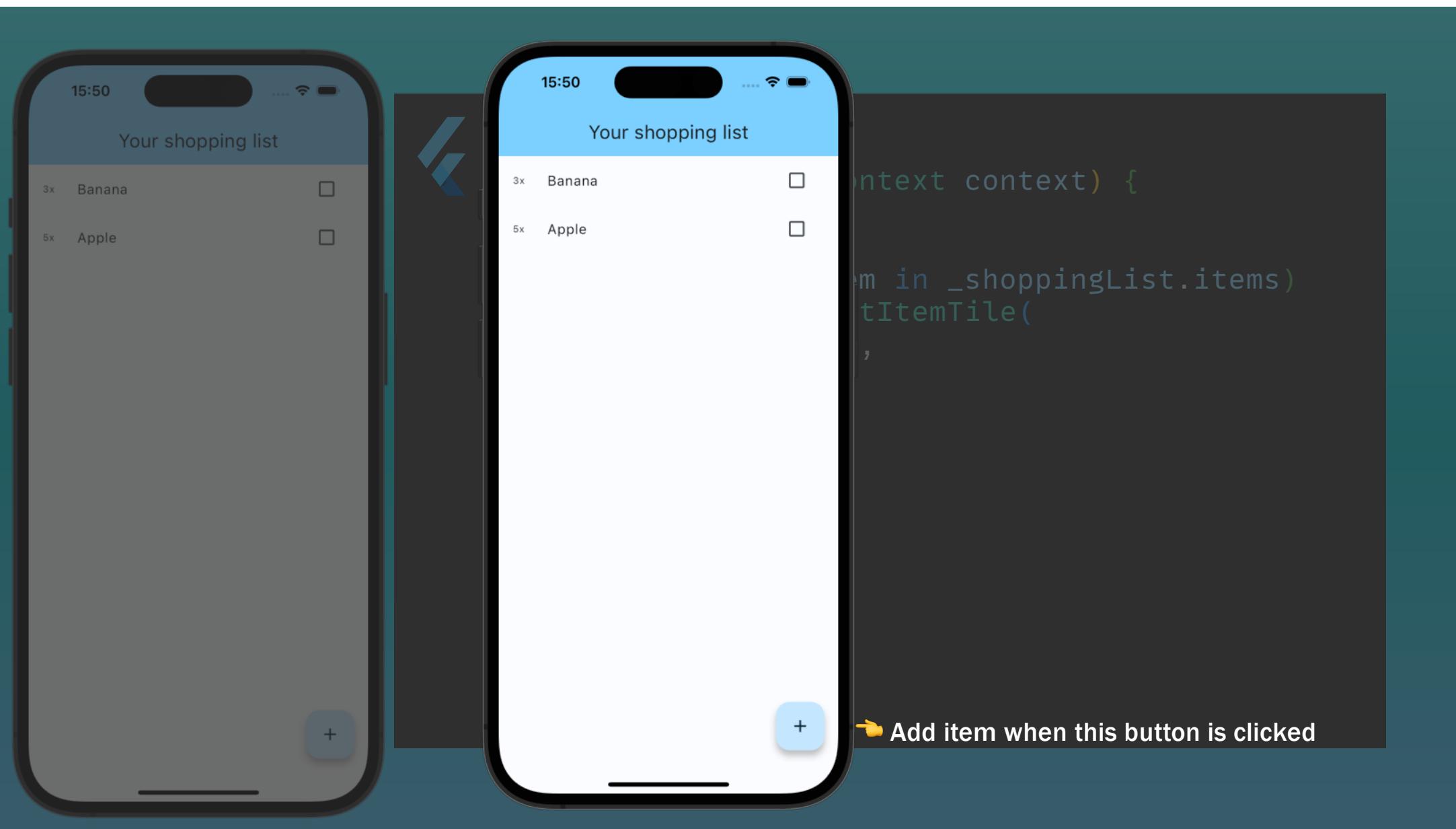


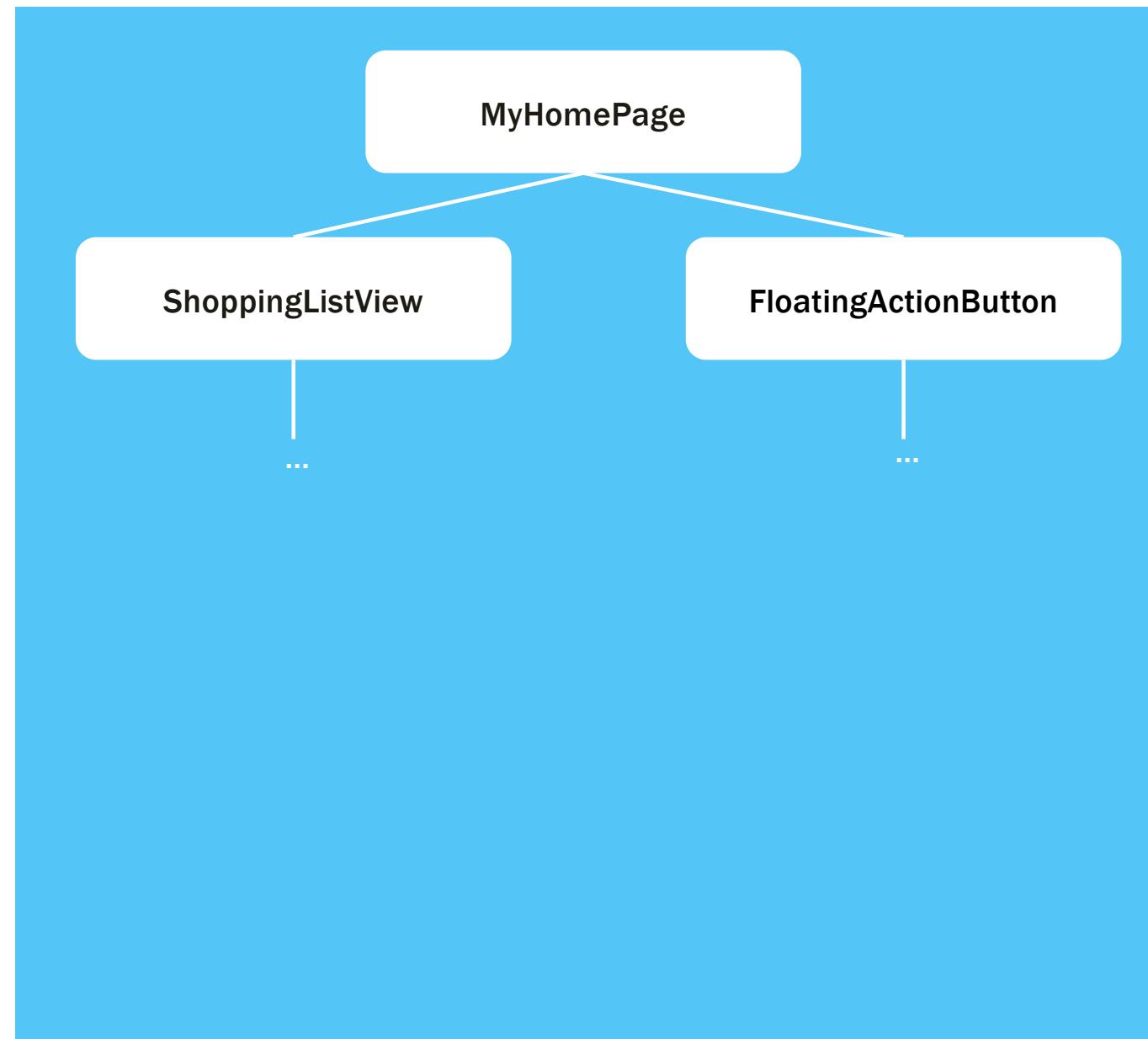
```
@override  
Widget build(BuildContext context) {  
  return ListView(  
    children: [  
      for (final item in _shoppingList.items)  
        _ShoppingListItemTile(  
          key: ValueKey(item.id)  
          item: item,  Uniquely identify each  
          ),  
        ],  
    );  
}
```

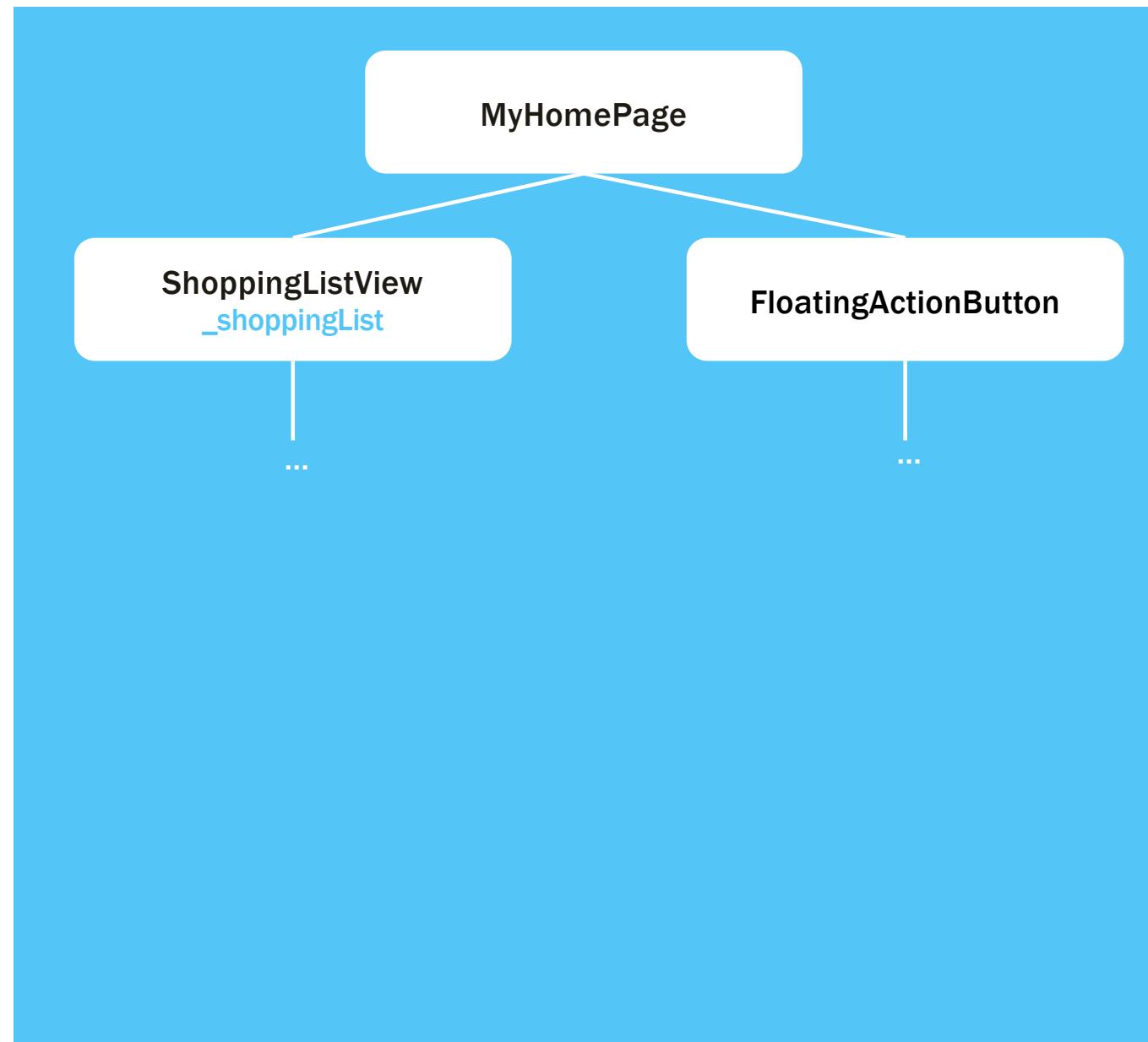


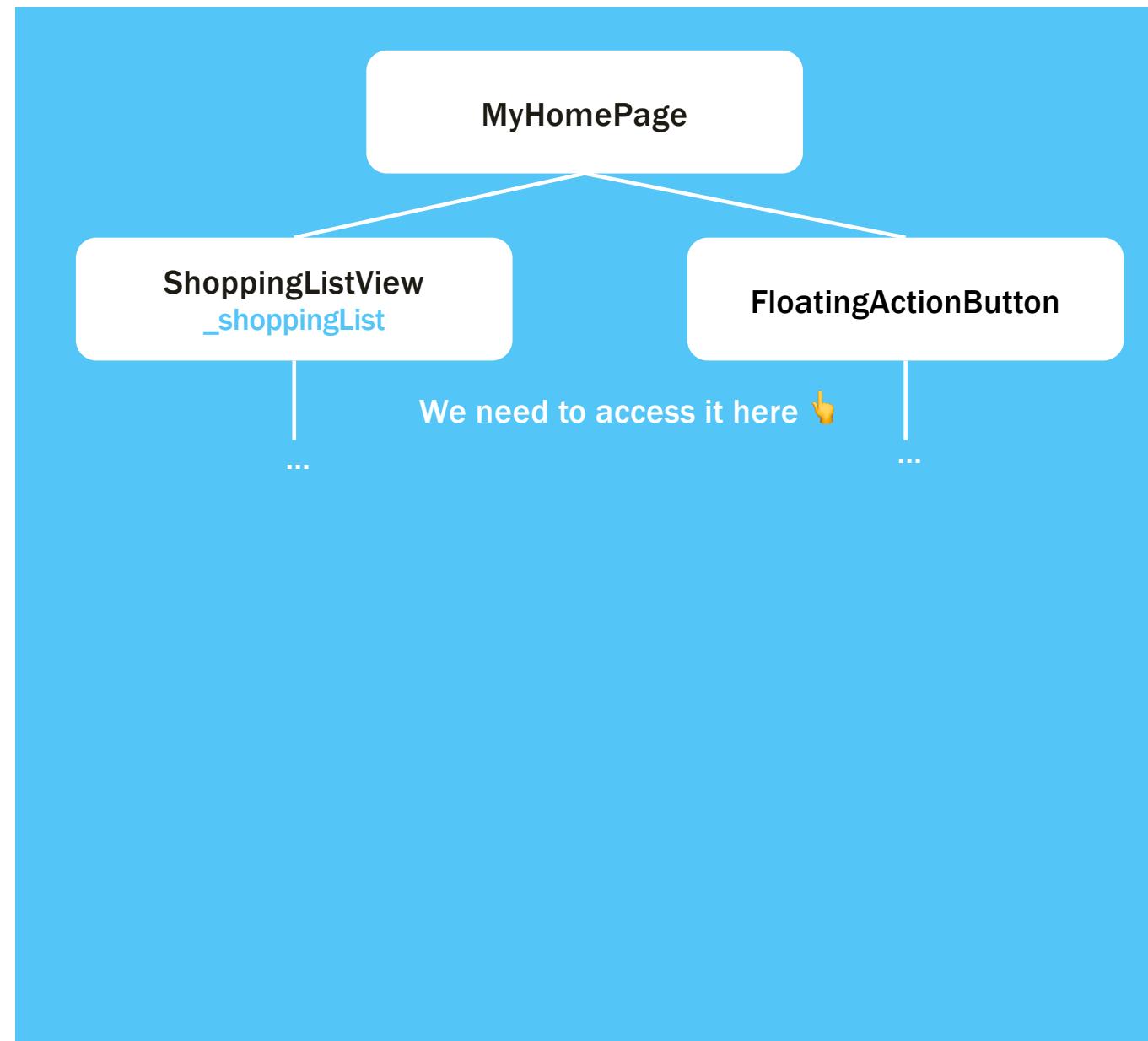
Adding Items

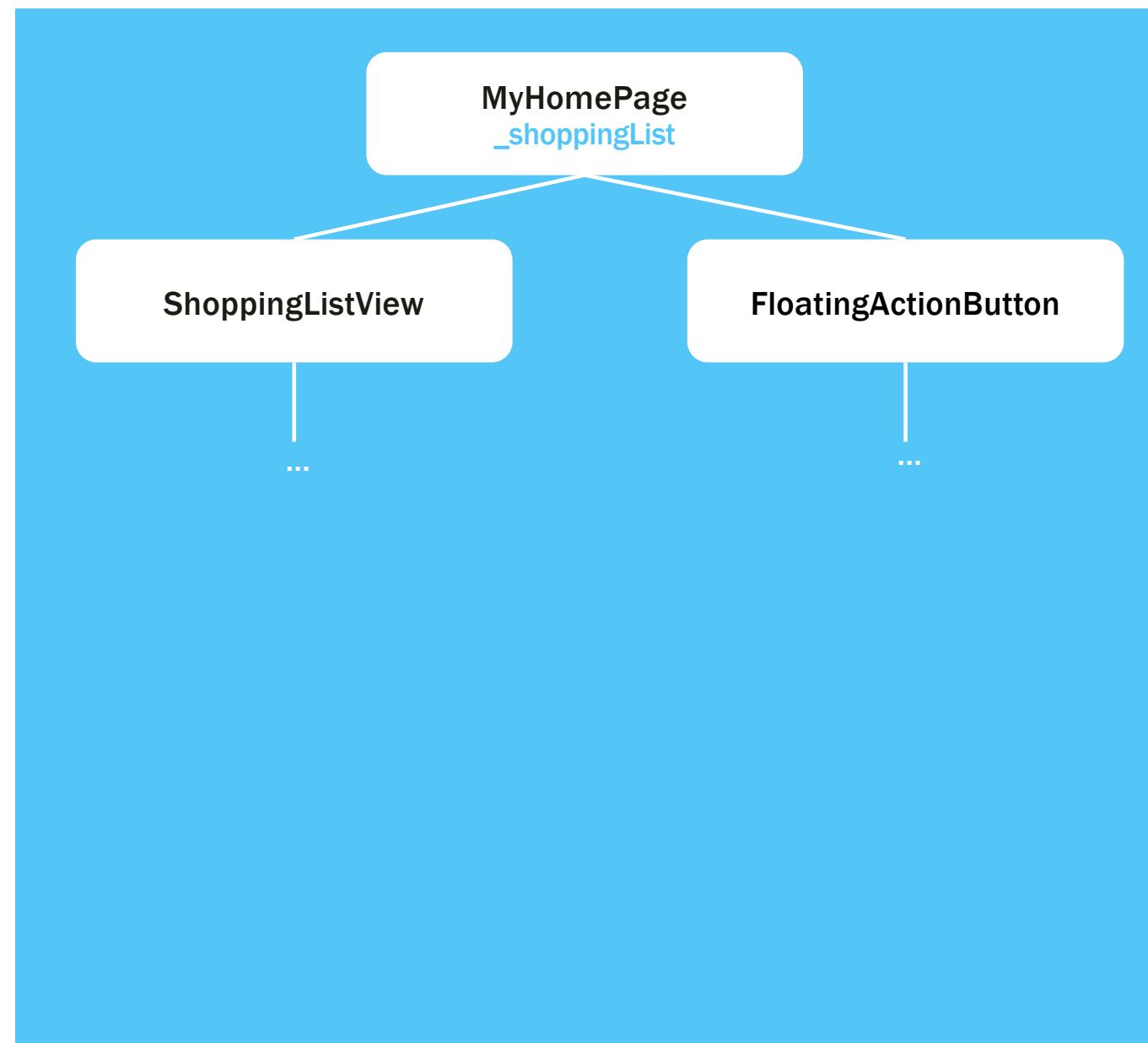
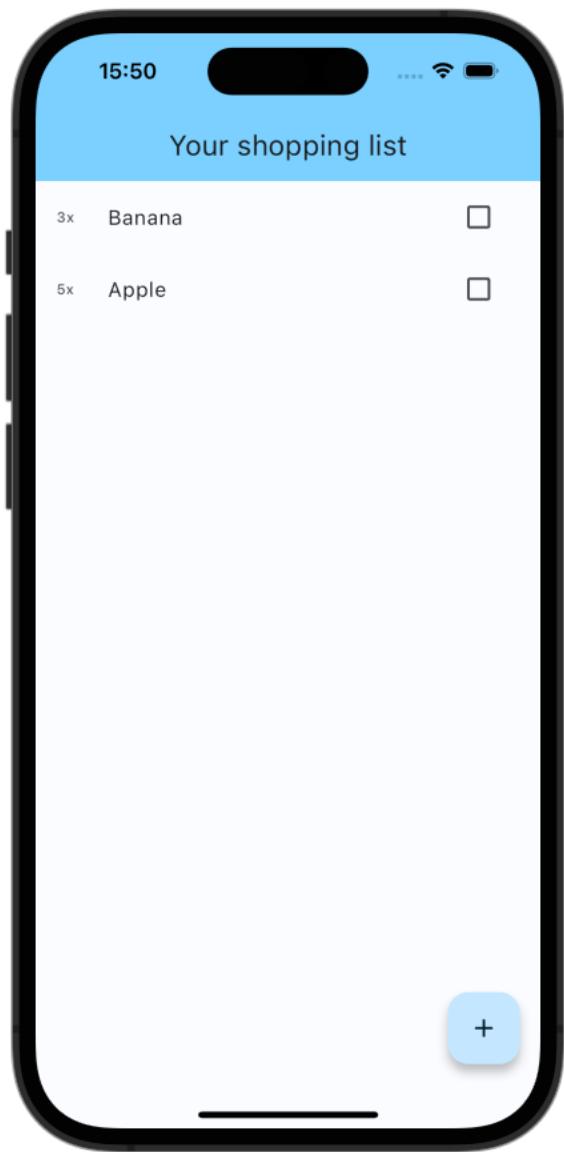
Shopping List App

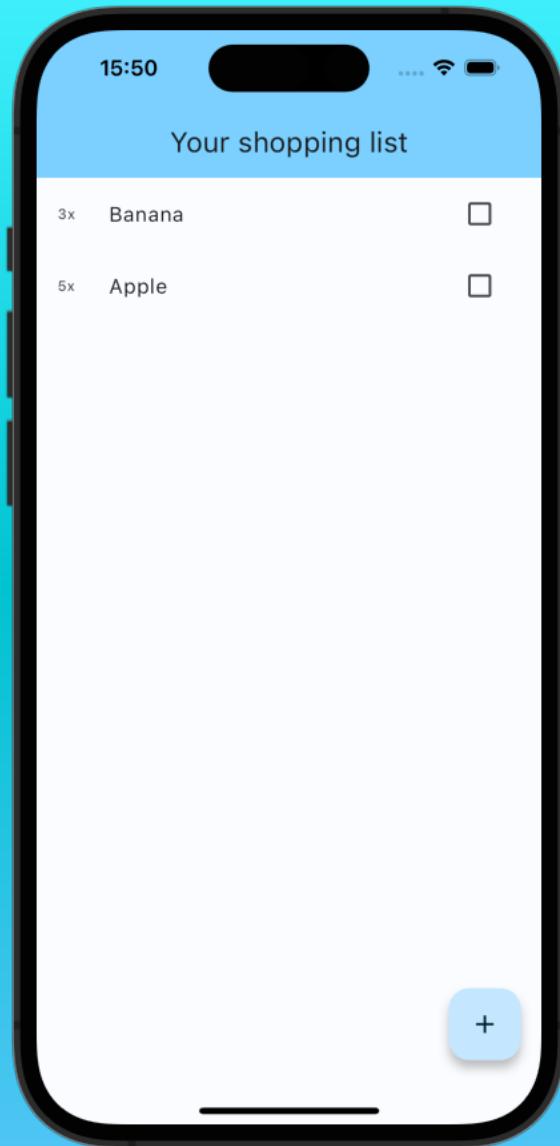






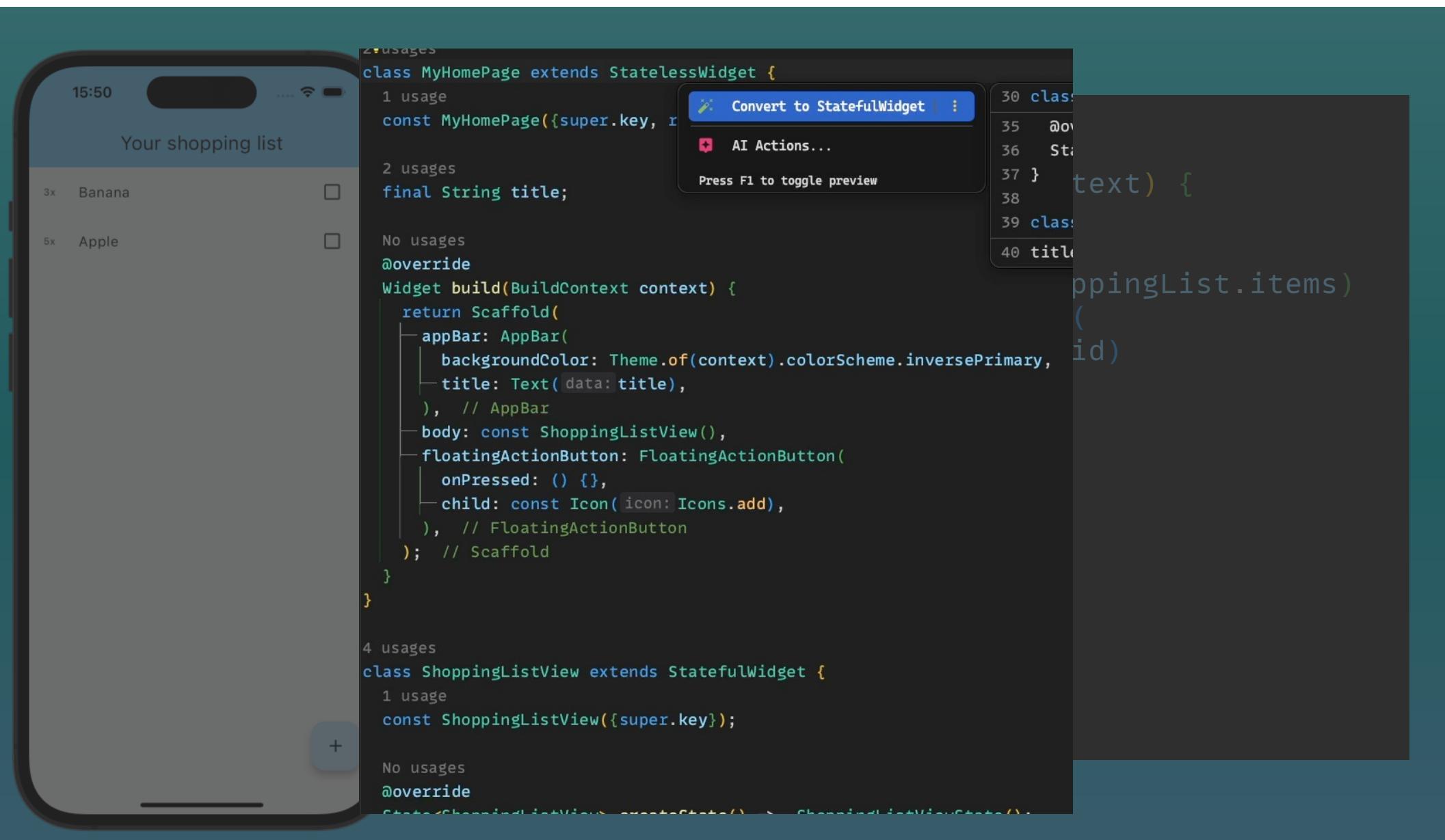


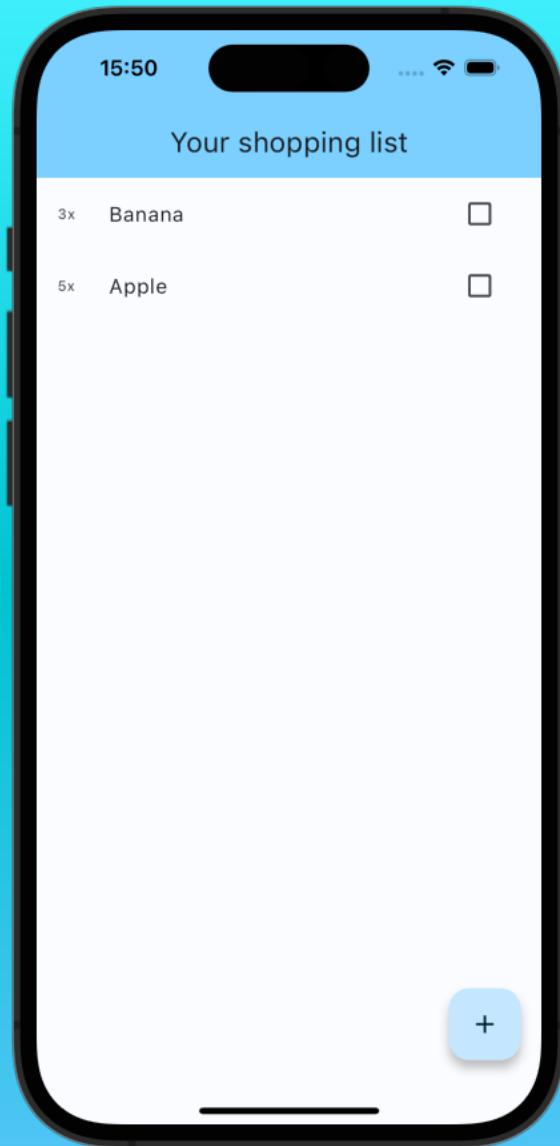




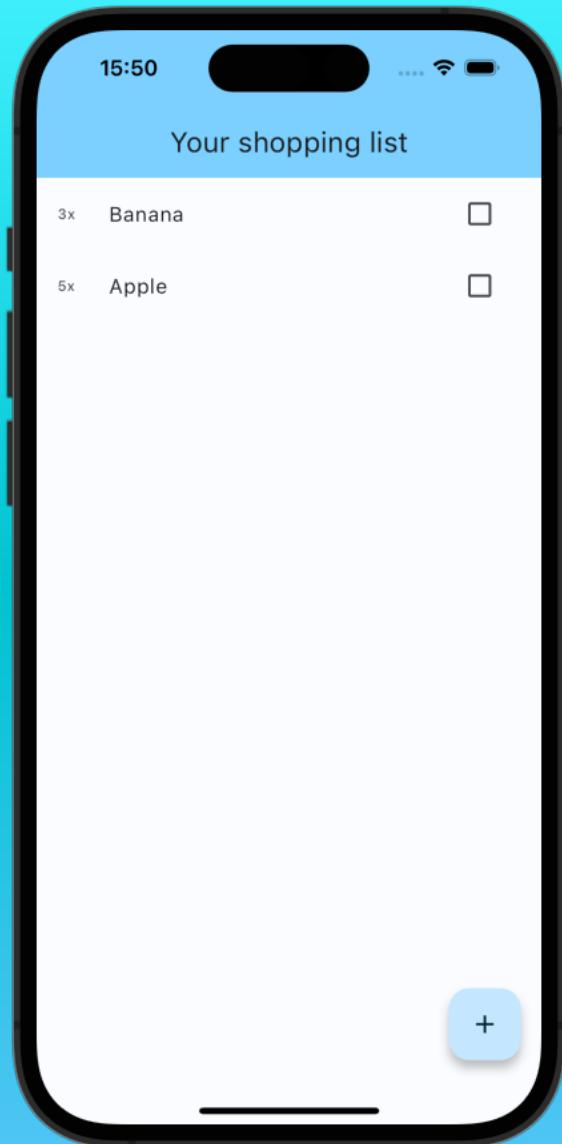
```
@override
Widget build(BuildContext context) {
    return ListView(
        children: [
            for (final item in _shoppingList.items)
                _ShoppingListItemTile(
                    key: ValueKey(item.id)
                    item: item,
                ),
        ],
    );
}
```

A dark grey rectangular panel is positioned to the right of the phone. It features the blue Flutter logo in the top-left corner. To its right is a block of Kotlin code. The code defines a method 'build' with a parameter 'BuildContext context'. It returns a 'ListView' with a single child: a 'for' loop that iterates over the '_shoppingList.items' list, creating a '_ShoppingListItemTile' for each item. The 'key' for each tile is generated by 'ValueKey(item.id)', and the 'item' is set to 'item'. The entire code block is enclosed in curly braces {}.





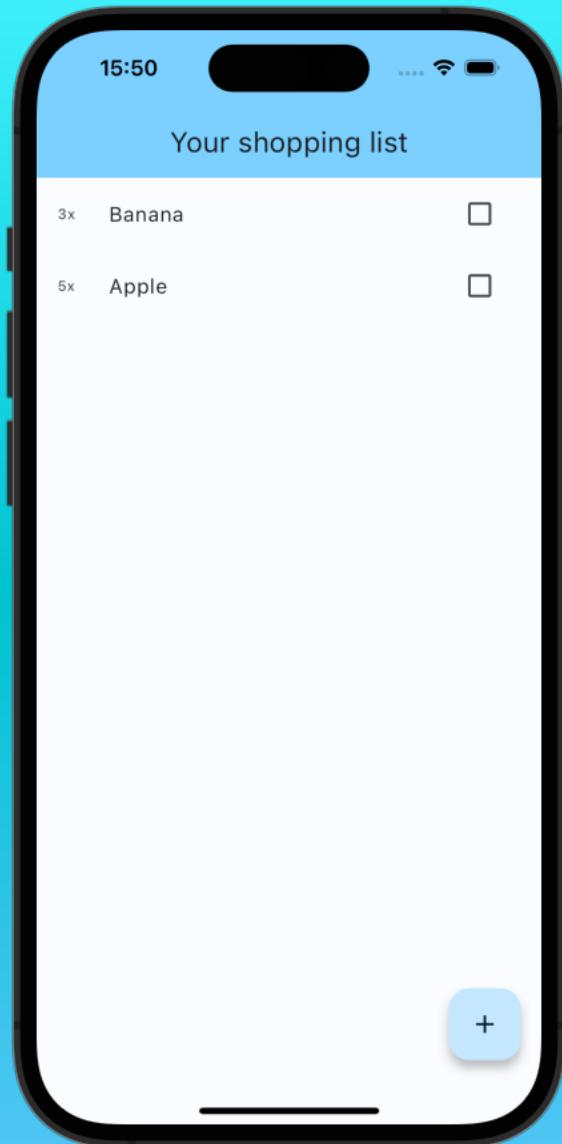
```
class _MyHomePageState extends State<MyHomePage>
{
  ShoppingList _shoppingList = ShoppingList()
    ..addItem(
      name: 'Banana',
      quantity: 3,
    )
    ..addItem(
      name: 'Apple',
      quantity: 5,
    );
  ...
}
```



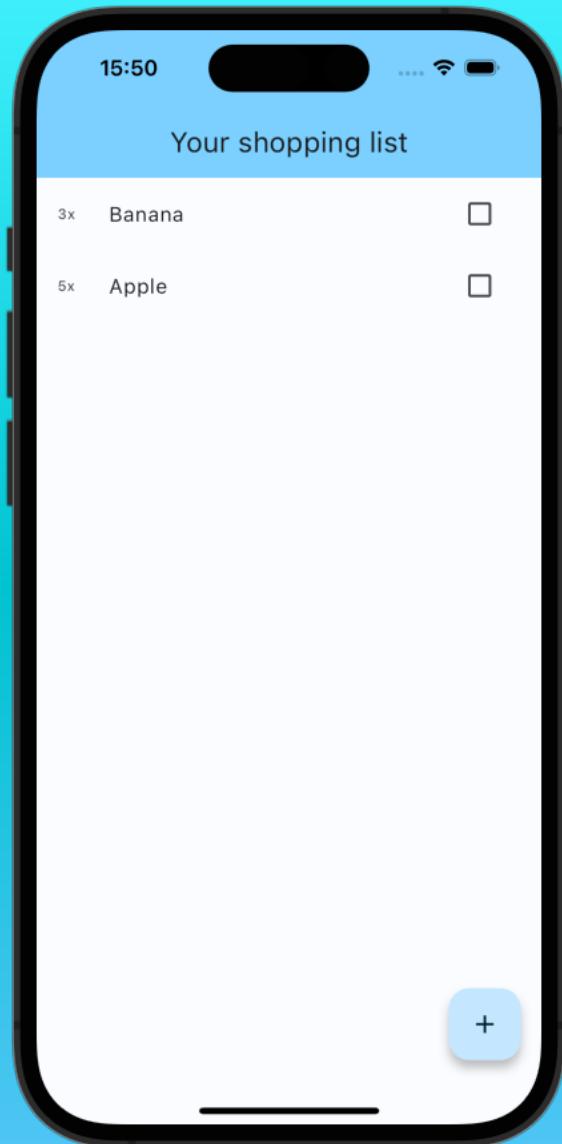
```
class ShoppingListView extends StatefulWidget {
  const ShoppingListView({super.key});

  @override
  State<ShoppingListView> createState() =>
  _ShoppingListViewState();
}

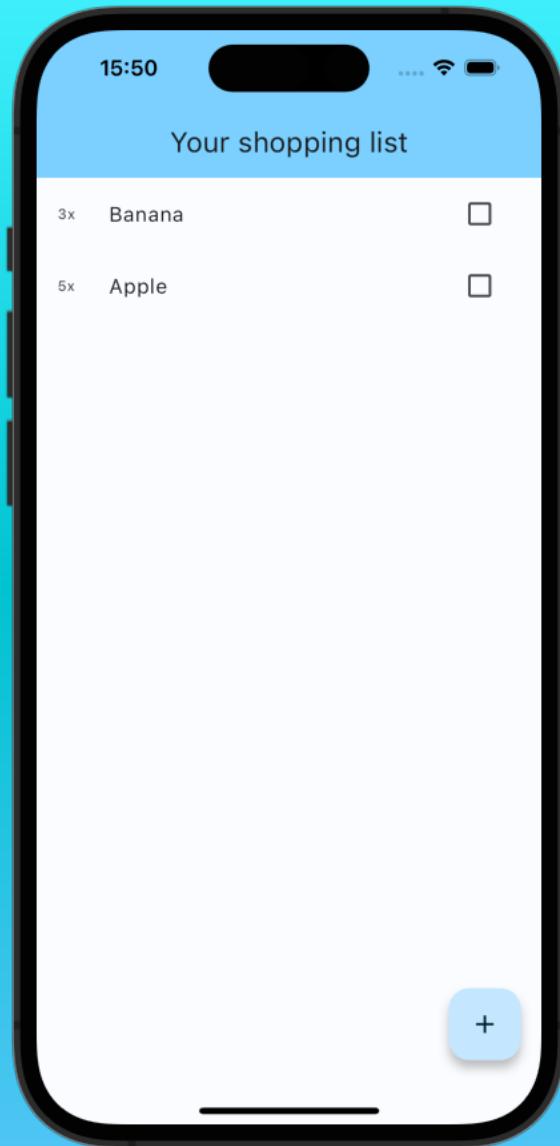
class _ShoppingListViewState extends State<ShoppingListView> {
  @override
  Widget build(BuildContext context) {
    return ListView(
      children: [
        for (final item in _shoppingList.items)
          _ShoppingListItemTile(
            key: ValueKey(item.id),
            item: item,
          ),
      ],
    );
  }
}
```



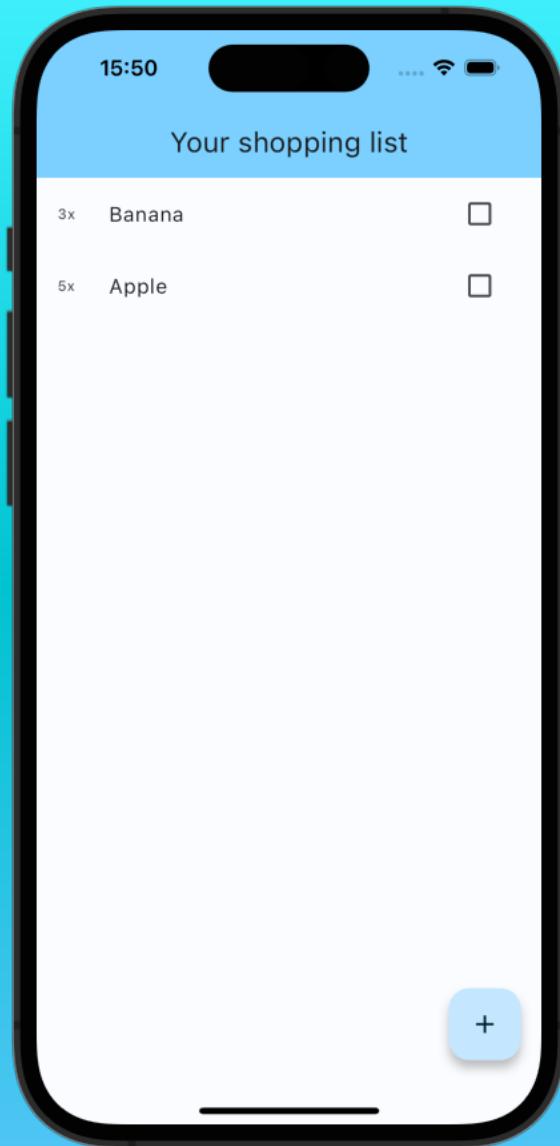
```
class ShoppingListView extends StatelessWidget {  
  const ShoppingListView({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return ListView(  
      children: [  
        for (final item in _shoppingList.items)  
          _ShoppingListItemTile(  
            key: ValueKey(item.id)  
            item: item,  
          ),  
      ],  
    );  
  }  
}
```



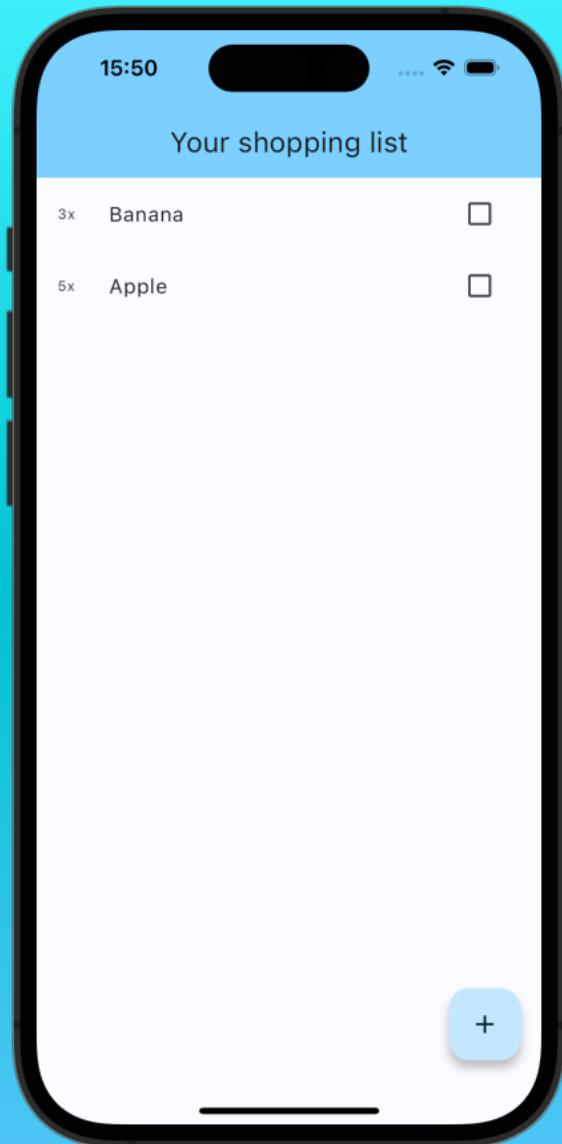
```
class ShoppingListView extends StatelessWidget {  
    const ShoppingListView({  
        required this.shoppingList,  
        super.key,  
    });  
  
    final ShoppingList shoppingList;  
    @override  
    Widget build(BuildContext context) {  
        return ListView(  
            children: [  
                for (final item in _shoppingList.items)  
                    _ShoppingListItemTile(  
                        key: ValueKey(item.id)  
                        item: item,  
                    ),  
            ],  
        );  
    }  
}
```



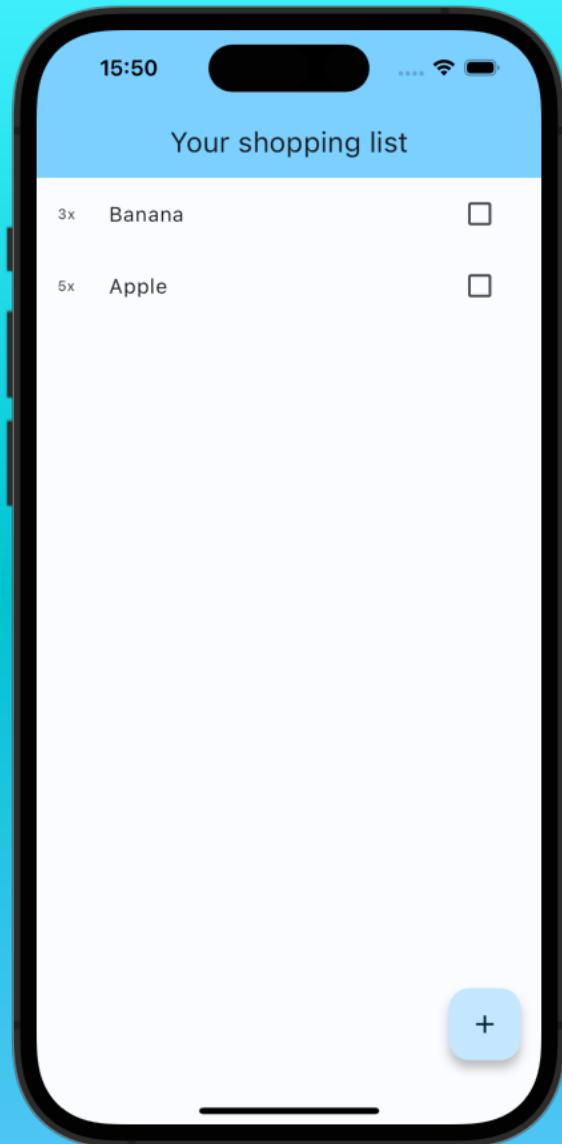
```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor:
Theme.of(context).colorScheme.inversePrimary,
        title: Text(widget.title),
      ),
      body: const ShoppingListView(),
      floatingActionButton: FloatingActionButton(
        onPressed: () {},
        child: const Icon(Icons.add),
      ),
    );
  }
}
```



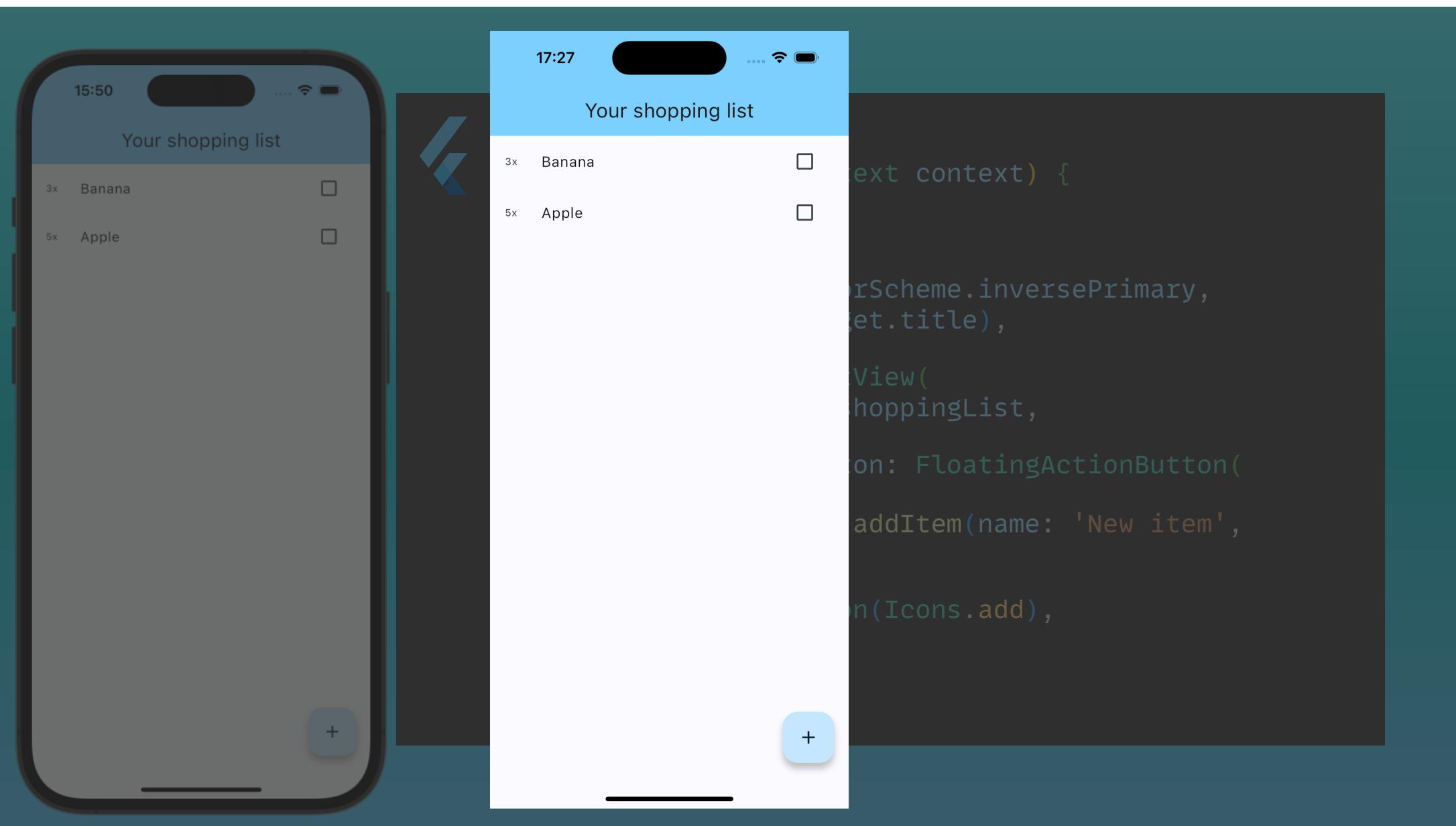
```
 @override  
Widget build(BuildContext context) {  
  return Scaffold(  
    appBar: AppBar(  
      backgroundColor:  
Theme.of(context).colorScheme.inversePrimary,  
      title: Text(widget.title),  
    ),  
    body: ShoppingListView(  
      shoppingList: _shoppingList,  
    ),  
    floatingActionButton: FloatingActionButton(  
      onPressed: () {},  
      child: const Icon(Icons.add),  
    ),  
  );  
}
```



```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor:
Theme.of(context).colorScheme.inversePrimary,
        title: Text(widget.title),
      ),
      body: ShoppingListView(
        shoppingList: _shoppingList,
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          _shoppingList.addItem(name: 'New item',
quantity: 1);
        },
        child: const Icon(Icons.add),
      );
    }
}
```



```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor:
Theme.of(context).colorScheme.inversePrimary,
        title: Text(widget.title),
      ),
      body: ShoppingListView(
        shoppingList: _shoppingList,
        floatingActionButton: FloatingActionButton(
          onPressed: () {
            setState(() {
              _shoppingList.addItem(name: 'New item',
quantity: 1);
            });
          },
          child: const Icon(Icons.add),
        ),
    );
}
```

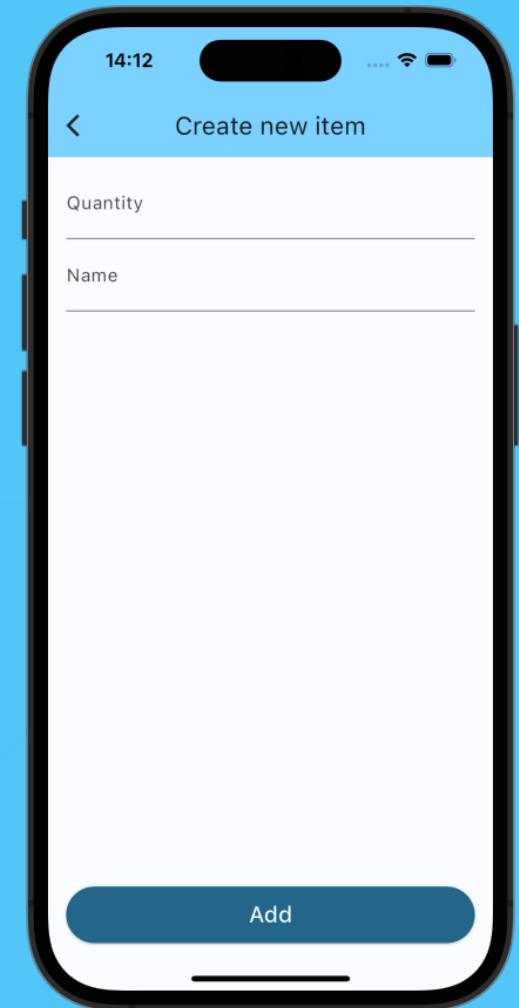


Creating a new page to add items

Shopping List App

Creating a new page to add items

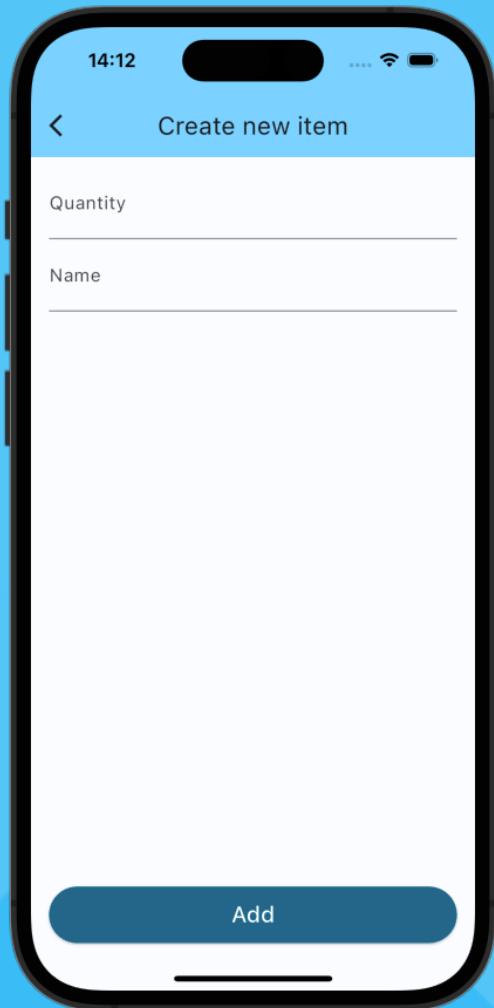
Shopping List App

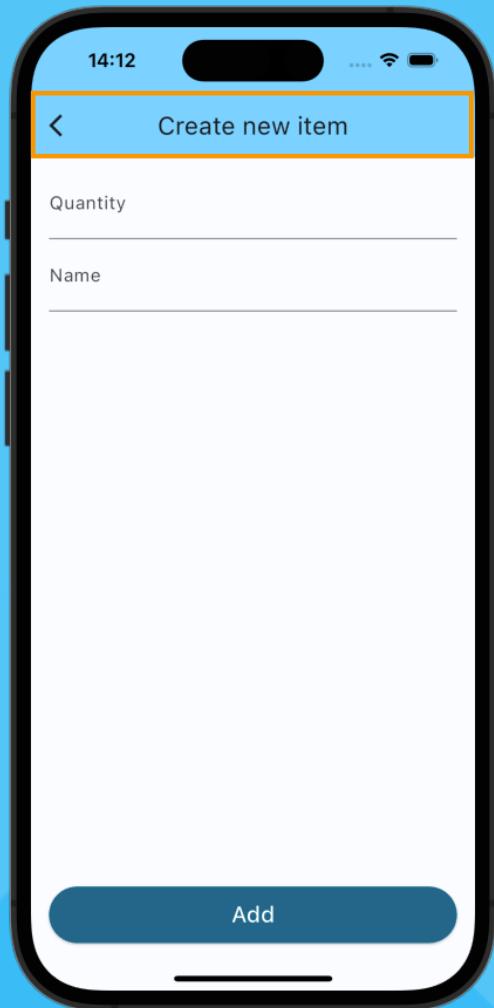


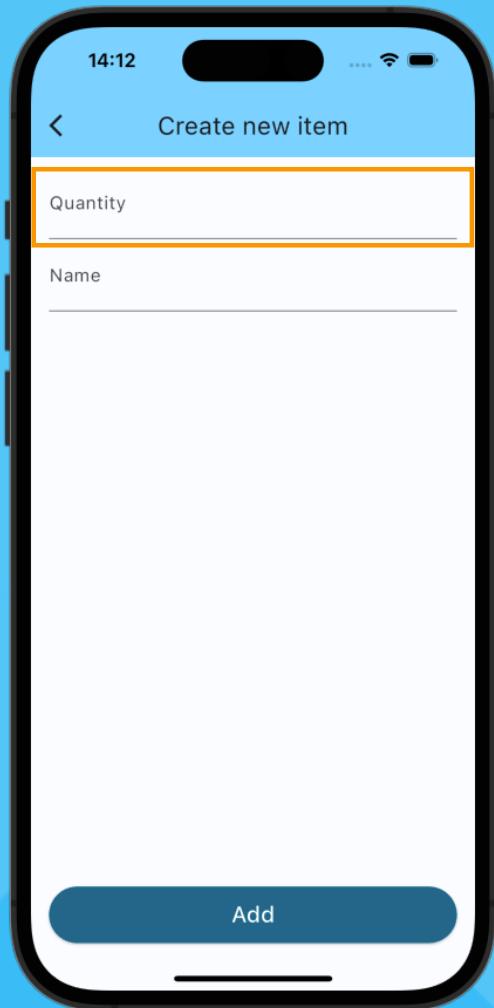


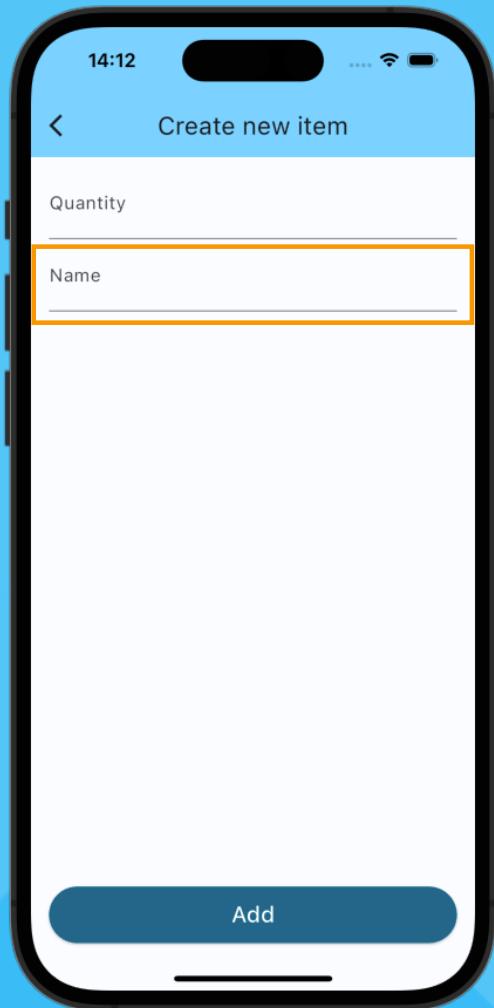
Breaking down the design

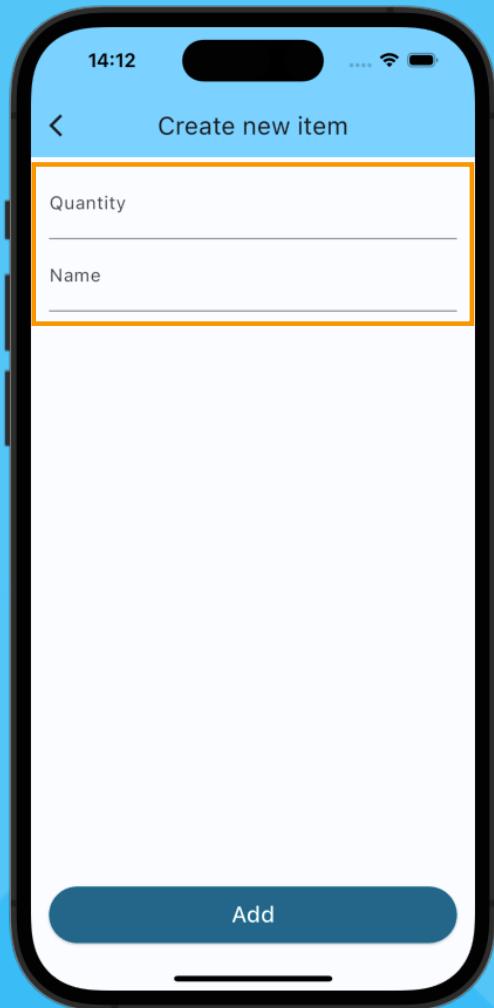
Shopping List App

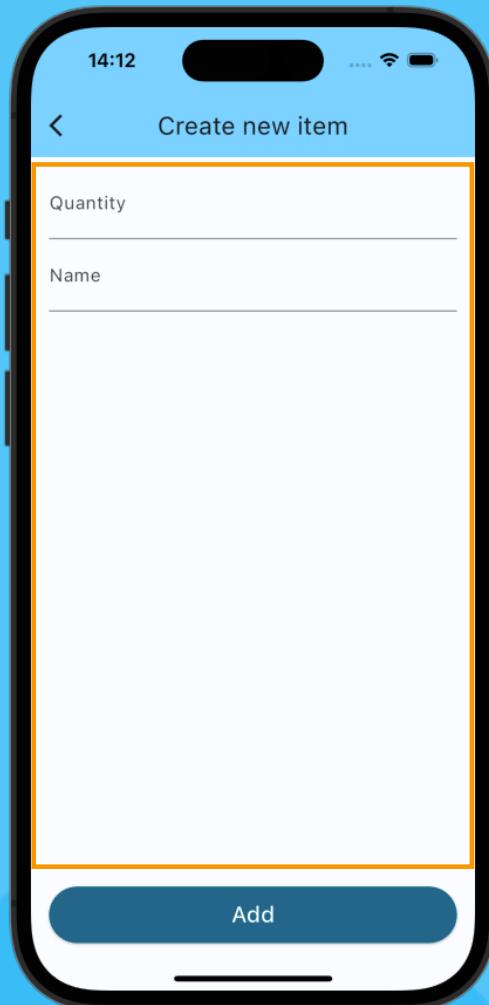


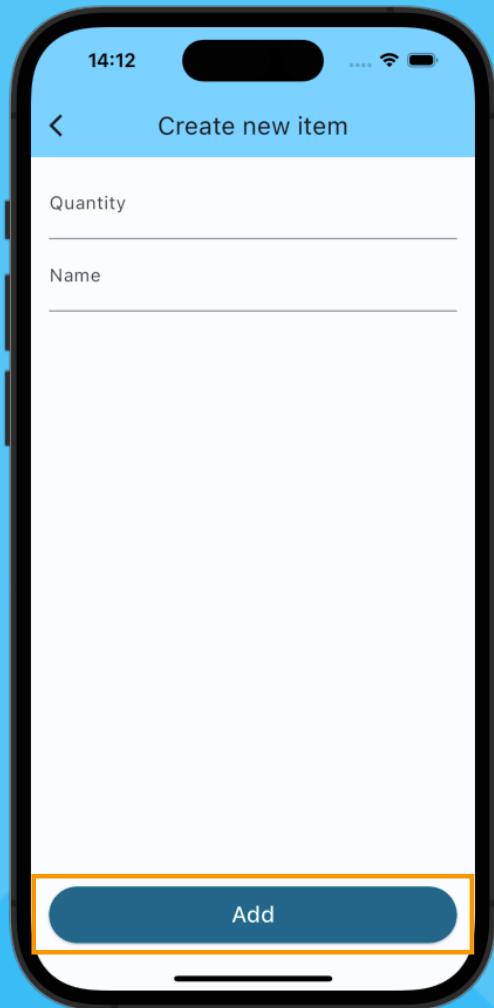






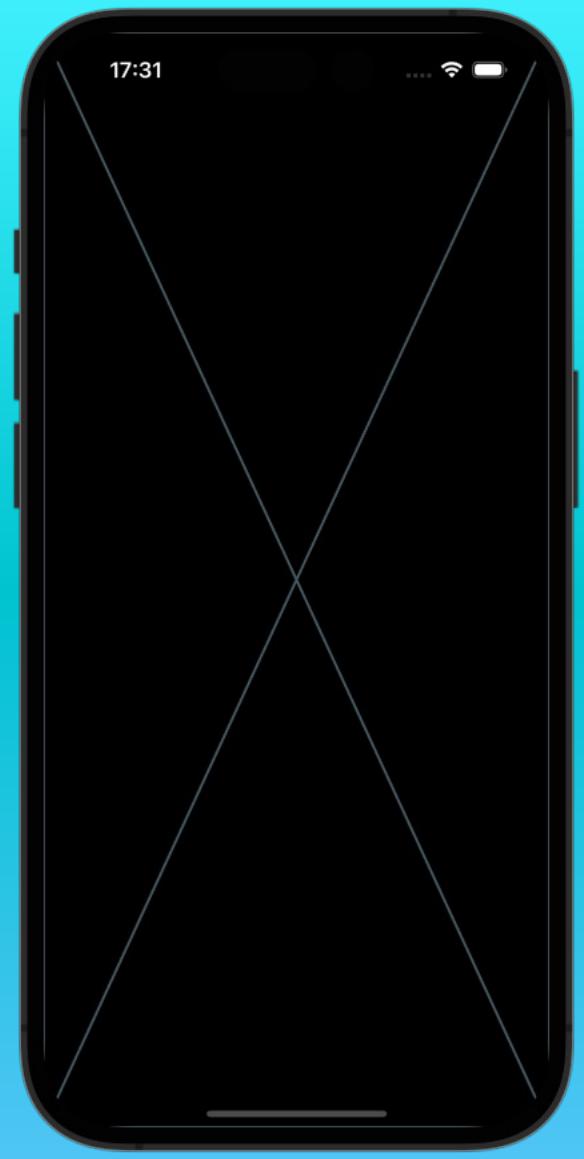






Implementing the page

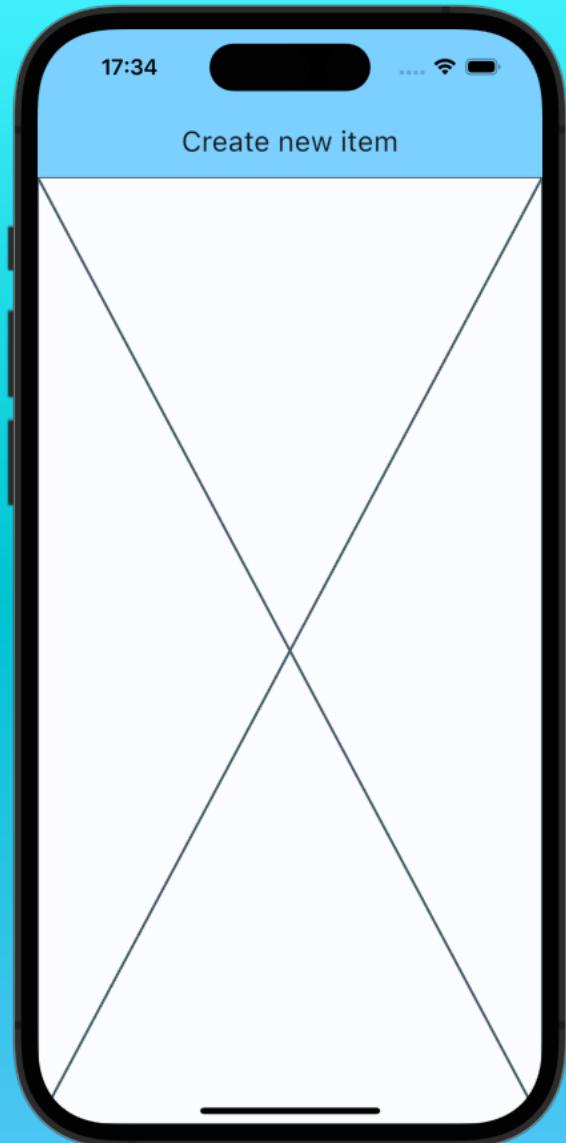
Shopping List App



17:31

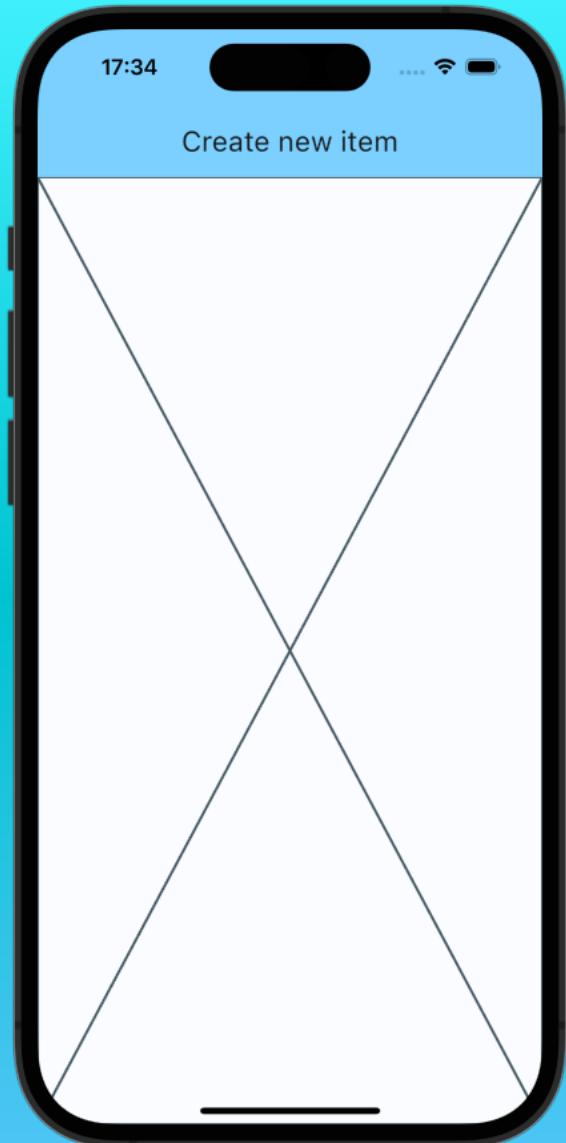


```
class CreateNewShoppingListWidgetItem extends  
StatelessWidget {  
  const CreateNewShoppingListWidgetItem({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return const Placeholder();  
  }  
}
```



```
class CreateNewShoppingListWidgetItem extends StatelessWidget {
  const CreateNewShoppingListWidgetItem({
    super.key,
    required this.shoppingList,
  });
  final ShoppingList shoppingList;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor:
        Theme.of(context).colorScheme.inversePrimary,
        title: const Text('Create new item'),
      ),
      body: _CreateNewItemForm(
        shoppingList: shoppingList,
      );
    }
}
```

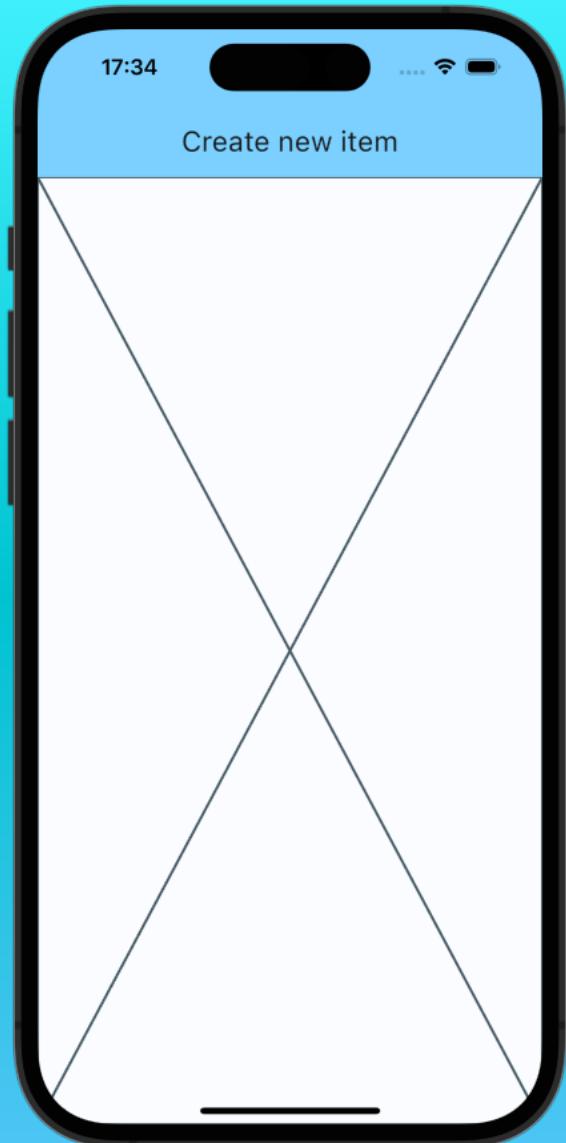


```
class _CreateNewItemForm extends StatefulWidget {
  const _CreateNewItemForm({
    super.key,
    required this.shoppingList,
  });

  final ShoppingList shoppingList;

  @override
  State<_CreateNewItemForm> createState() =>
  _CreateNewItemFormState();
}

class _CreateNewItemFormState extends State<_CreateNewItemForm> {
  @override
  Widget build(BuildContext context) {
    return const Placeholder();
}
```



```
class _CreateNewItemFormState extends State<_CreateNewItemForm> {
  @override
  Widget build(BuildContext context) {
    return Form(
      child: ,
    );
  }
}
```



17:38

Create new item



```
class _CreateNewItemFormState extends  
State<_CreateNewItemForm> {  
  @override  
  Widget build(BuildContext context) {  
    return Form(  
      child: Column(  
        children: [],  
      ),  
    );  
  }  
}
```

Column



“A widget that displays its children in a vertical array.”

Row

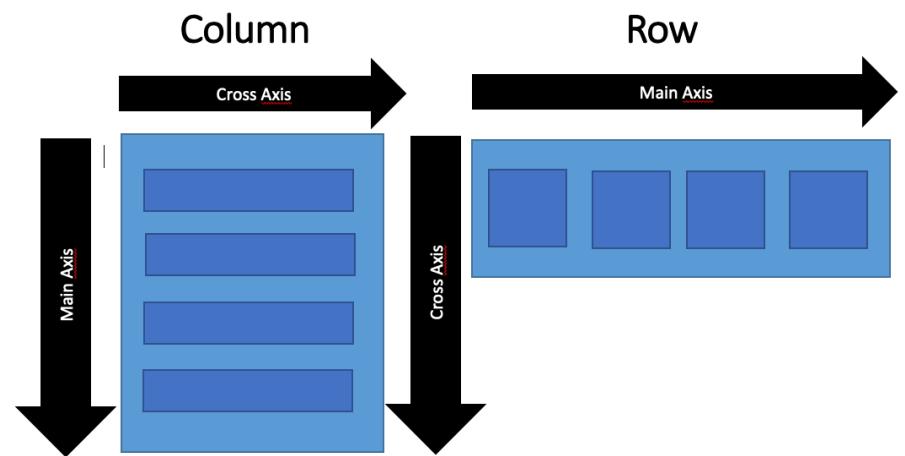


“A widget that displays its children in a horizontal array.“

Column/Row



- **children:** “The widgets below this widget in the tree.”
- **mainAxisAlignment:** “How the children should be placed along the main axis.”
- **mainAxisSize:** “How much space should be occupied in the main axis.”
- **crossAxisAlignment:** “How the children should be placed along the cross axis.”
- **see all properties [here](#)**



Column/Row vs. Flexbox

mainAxisAlignment

- start
- end
- center
- spaceBetween
- spaceAround
- spaceEvenly

justify-content

- start
- end
- center
- space-between
- space-around
- space-evenly

Column/Row vs. Flexbox

crossAxisAlignment

- start
- end
- center
- stretch
- baseline

align-items

- start
- end
- center
- stretch
- baseline

Column/Row vs. Flexbox

Expanded

- “A widget that expands a child of Row, Column, or Flex so that the child fills the available space.”

flex

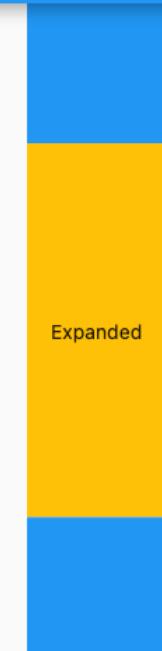
- “flex CSS shorthand property sets how a flex item will grow or shrink to fit the space available in its flex container.”

Expanded

Expanded Row Sample



Expanded Column Sample

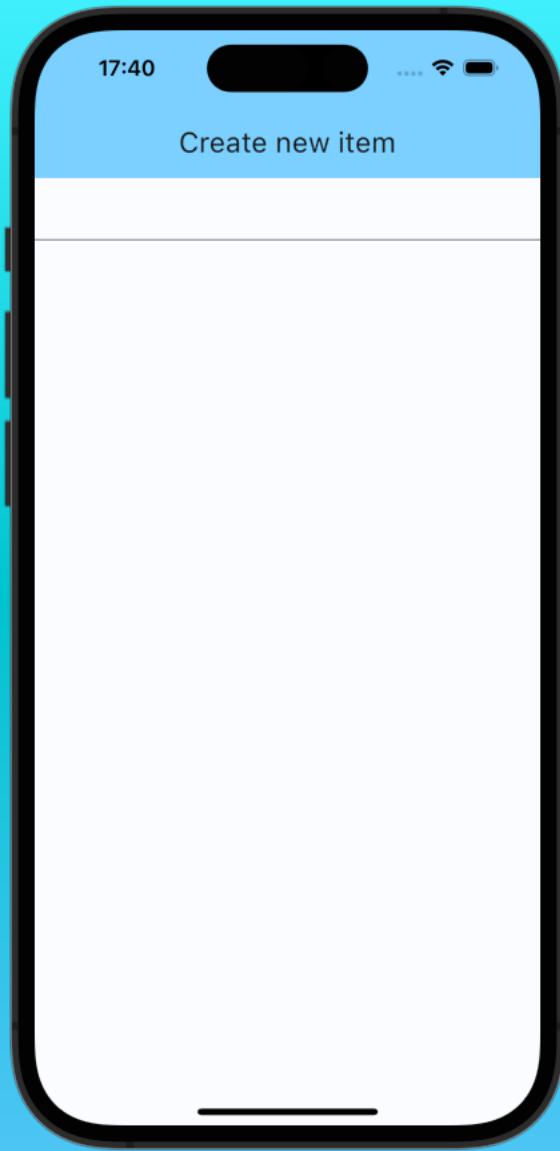


17:38

Create new item



```
class _CreateNewItemFormState extends  
State<_CreateNewItemForm> {  
  @override  
  Widget build(BuildContext context) {  
    return Form(  
      child: Column(  
        children: [],  
      ),  
    );  
  }  
}
```



```
class _CreateNewItemFormState extends State<_CreateNewItemForm> {
  @override
  Widget build(BuildContext context) {
    return Form(
      child: Column(
        children: [
          TextFormField(),
        ],
      );
    }
}
```

TextField

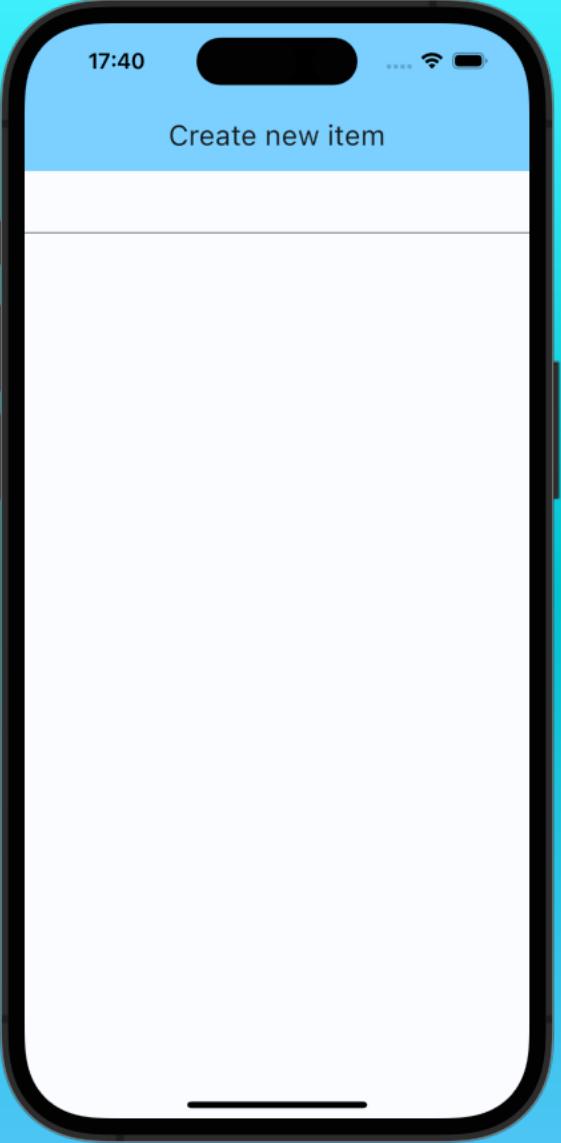


“A **FormField** that contains a **TextField**. This is a convenience widget that wraps a **TextField** widget in a **FormField**.“

TextField



- **decoration:** “The decoration to show around the text field.“
- **controller:** “Controls the text being edited.“
- **validator:** “An optional method that validates the input. Returns an error string to display if the input is invalid, or null otherwise.“
- see all properties [here](#)



17:40

Create new item



```
class _CreateNewItemFormState extends  
State<_CreateNewItemForm> {  
  @override  
  Widget build(BuildContext context) {  
    return Form(  
      child: Column(  
        children: [  
          TextFormField(),  
        ],  
      ),  
    );  
  }  
}
```

17:41

Create new item

Name



```
class _CreateNewItemFormState extends  
State<_CreateNewItemForm> {  
  @override  
  Widget build(BuildContext context) {  
    return Form(  
      child: Column(  
        children: [  
          TextFormField(  
            decoration: InputDecoration(  
              labelText: 'Name',  
            ),  
            ),  
            ],  
          );  
    }  
}
```

Padding



“A widget that insets its child by the given padding.“

Padding



- **child:** “The widget below this widget in the tree.“
- **padding:** “The amount of space by which to inset the child.“
- **see all properties [here](#)**

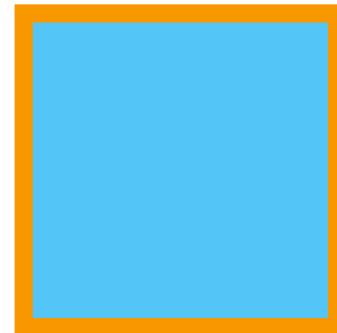
Understanding EdgeInsets

Shopping List App

EdgeInsets.all



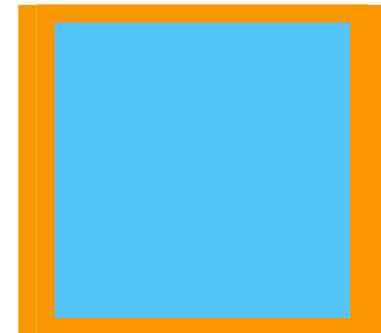
```
const EdgeInsets.all(8.0);
```



EdgeInsets.symmetric

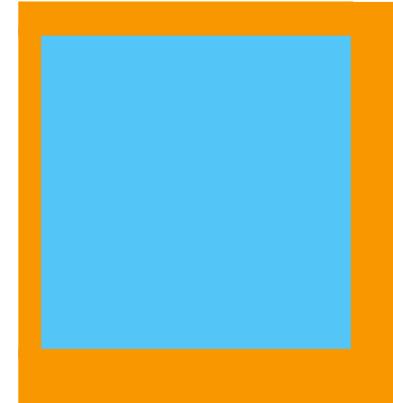


```
const EdgeInsets.symmetric(  
    horizontal: 16,  
    vertical: 8,  
);
```



EdgeInsets.fromLTRB

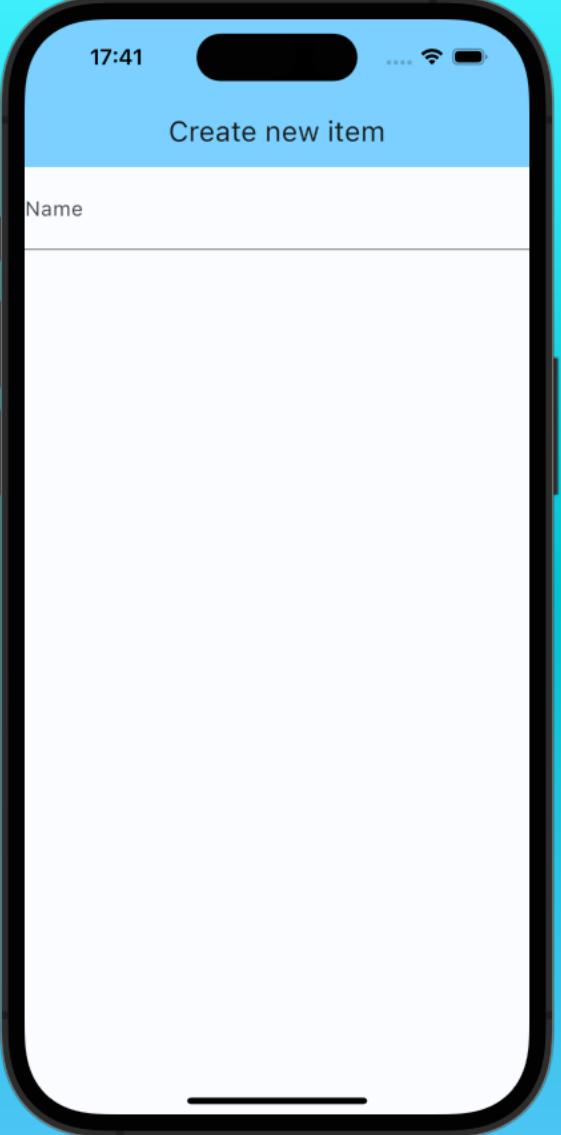
```
const EdgeInsets.fromLTRB(  
  10,  
  15,  
  20,  
  25,  
);
```



EdgeInsets.only



```
const EdgeInsets.only(left: 10.0);
```



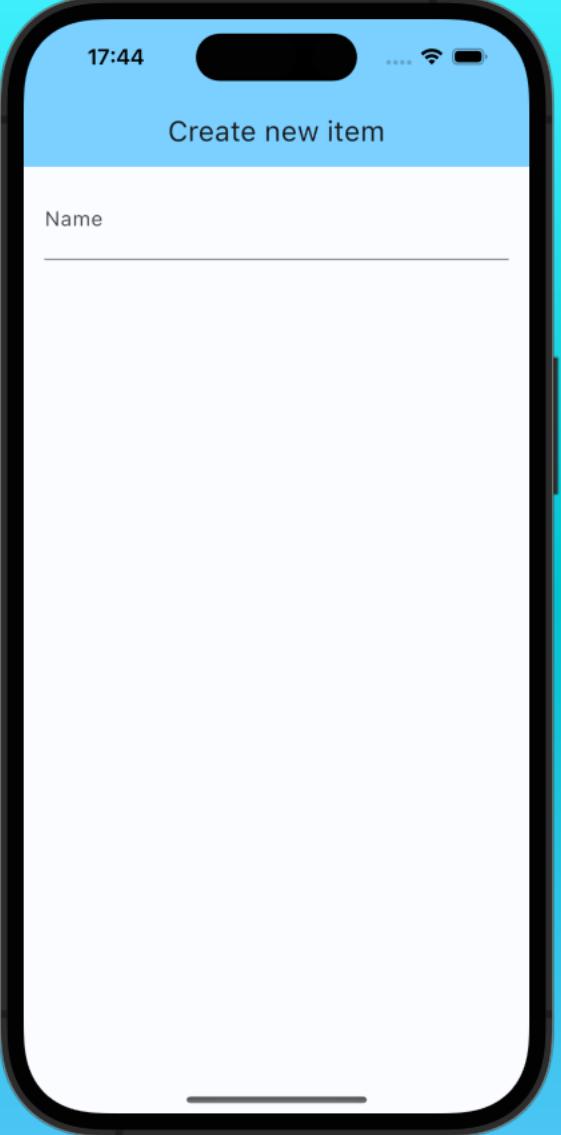
17:41

Create new item

Name



```
class _CreateNewItemFormState extends  
State<_CreateNewItemForm> {  
  @override  
  Widget build(BuildContext context) {  
    return Form(  
      child: Column(  
        children: [  
          TextFormField(  
            decoration: InputDecoration(  
              labelText: 'Name',  
            ),  
          ),  
        ],  
      ),  
    );  
  }  
}
```



17:44

Create new item

Name



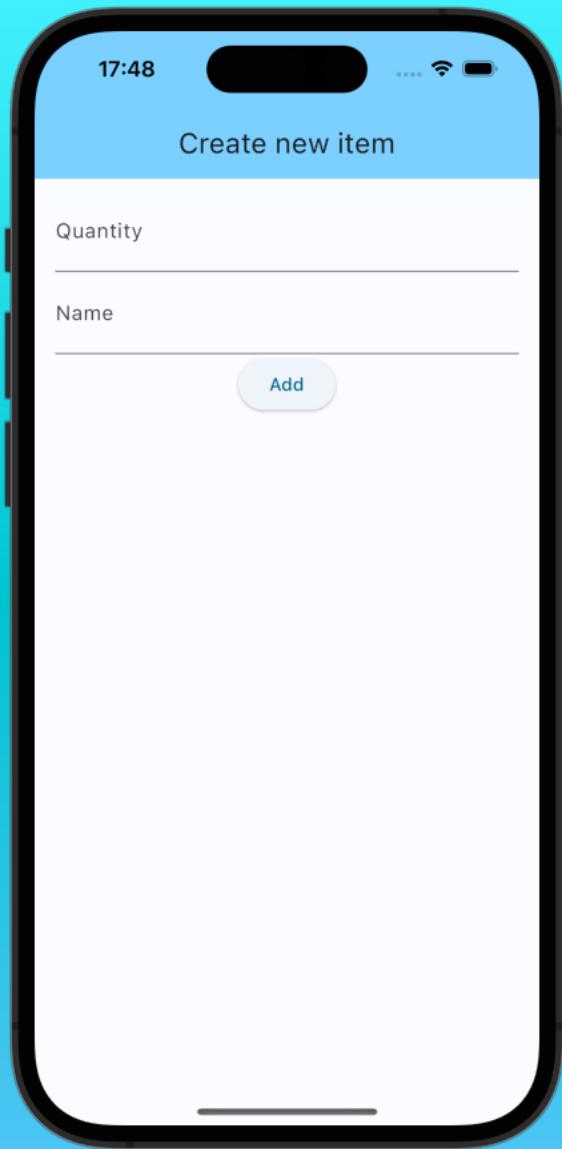
```
@override
Widget build(BuildContext context) {
  return Padding(
    padding: const EdgeInsets.symmetric(
      horizontal: 16,
      vertical: 8,
    ),
    child: Form(
      child: Column(
        children: [
          TextFormField(
            decoration: InputDecoration(
              labelText: 'Name',
            ),
          ),
        ],
      ),
    );
}
```

17:44

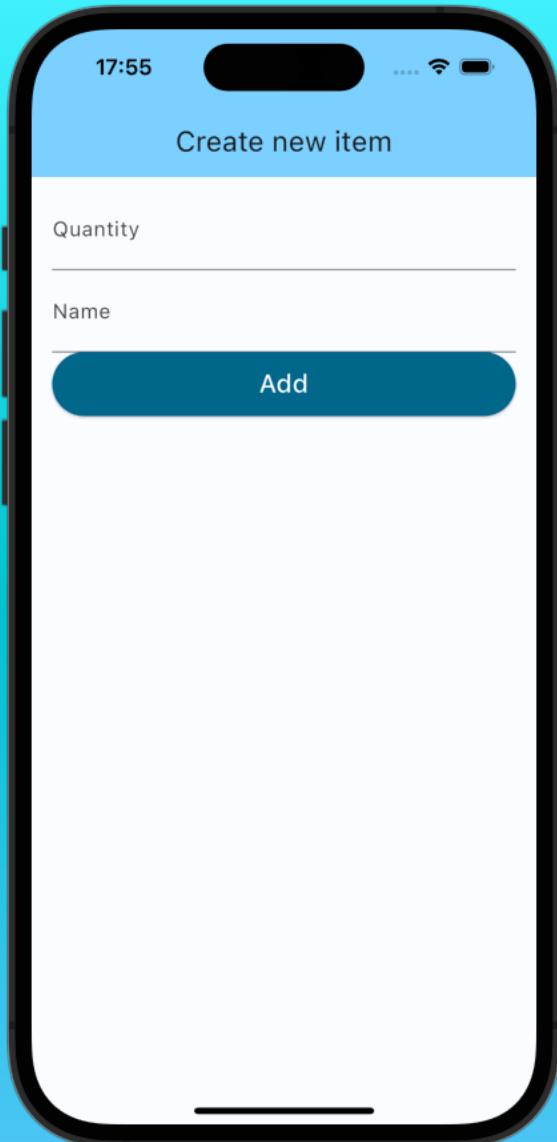
Create new item

Name

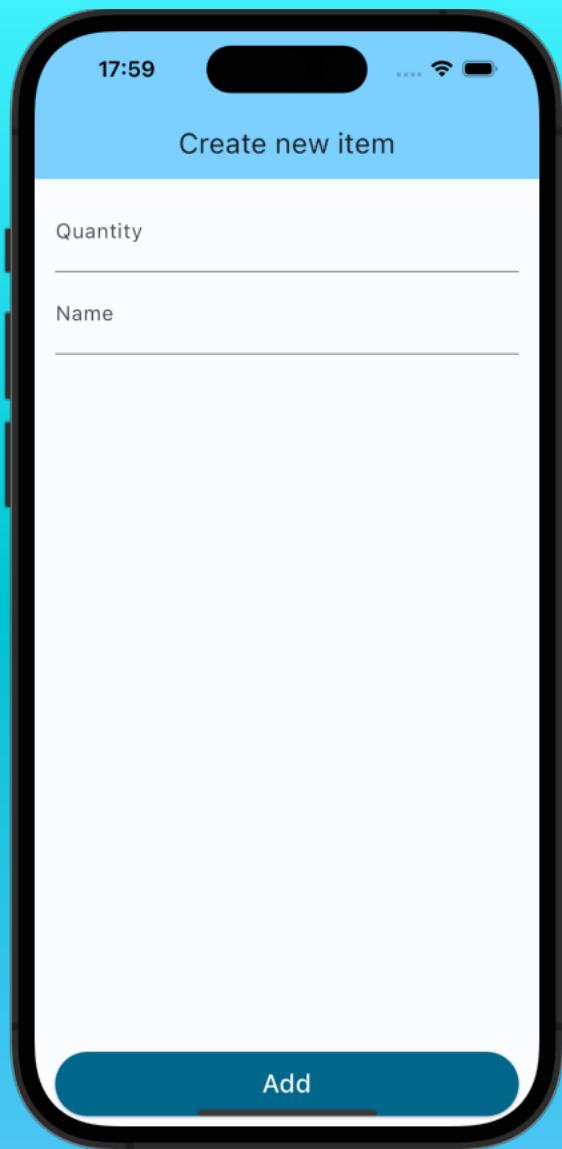
```
No usages
@Override
Widget build(BuildContext context) {
    return Form(...); // Form
}
```



```
[  
  TextFormField(  
    decoration: InputDecoration(  
      labelText: 'Quantity',  
    ),  
  ),  
  TextFormField(  
    decoration: InputDecoration(  
      labelText: 'Name',  
    ),  
  ),  
  ElevatedButton(  
    onPressed: () {},  
    child: const Text('Add'),  
  ),  
,
```



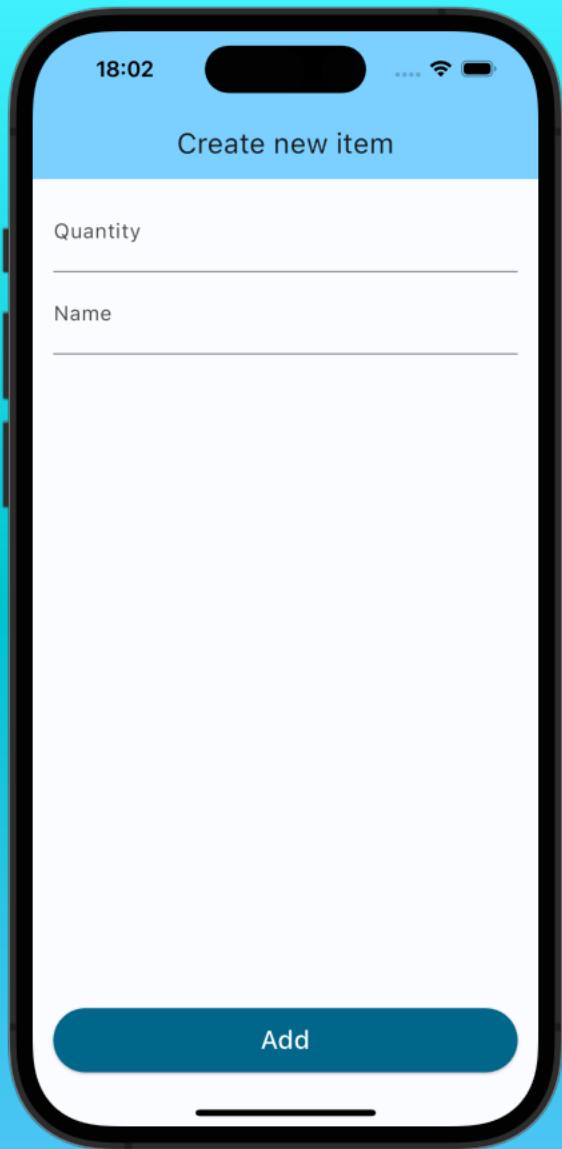
```
  ElevatedButton(  
    style: ElevatedButton.styleFrom(  
      backgroundColor:  
        Theme.of(context).colorScheme.primary,  
      foregroundColor:  
        Theme.of(context).colorScheme.onPrimary,  
      minimumSize: const Size.fromHeight(50.0),  
    ),  
    onPressed: () {},  
    child: const Text(  
      'Add',  
      style: TextStyle(fontSize: 20),  
    ),  
  ),
```



```
[  
  Expanded(  
    child: Column(  
      children: [  
        TextFormField(  
          ...  
        ),  
        TextFormField(  
          ...  
        ),  
        ],  
      ),  
      ElevatedButton(  
        ...  
      ),  
    ]
```

SafeArea

“A widget that insets its child by sufficient padding to avoid intrusions by the operating system.”



```
return SafeArea(  
    child: ...  
);
```

A dark gray rectangular area on the right side of the image contains the Flutter logo icon (a stylized blue and white "K") followed by a snippet of Dart code. The code is a single line that returns a `SafeArea` widget with a child. The entire dark gray area is set against the same solid cyan background as the phone.

Capturing Input

Shopping List App

Capturing Input

- Two viable approaches

Capturing Input

- Two viable approaches
 1. Using a state variable and setting it in the `onChanged` callback of the `TextField`
 2. Using a `TextEditingController`

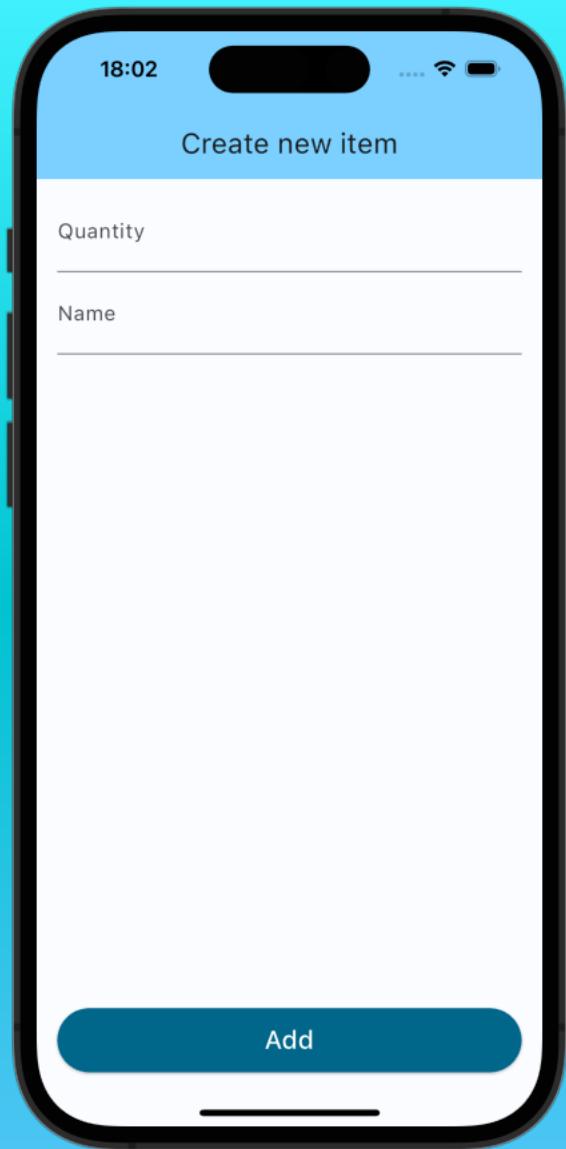
Capturing Input

State variables

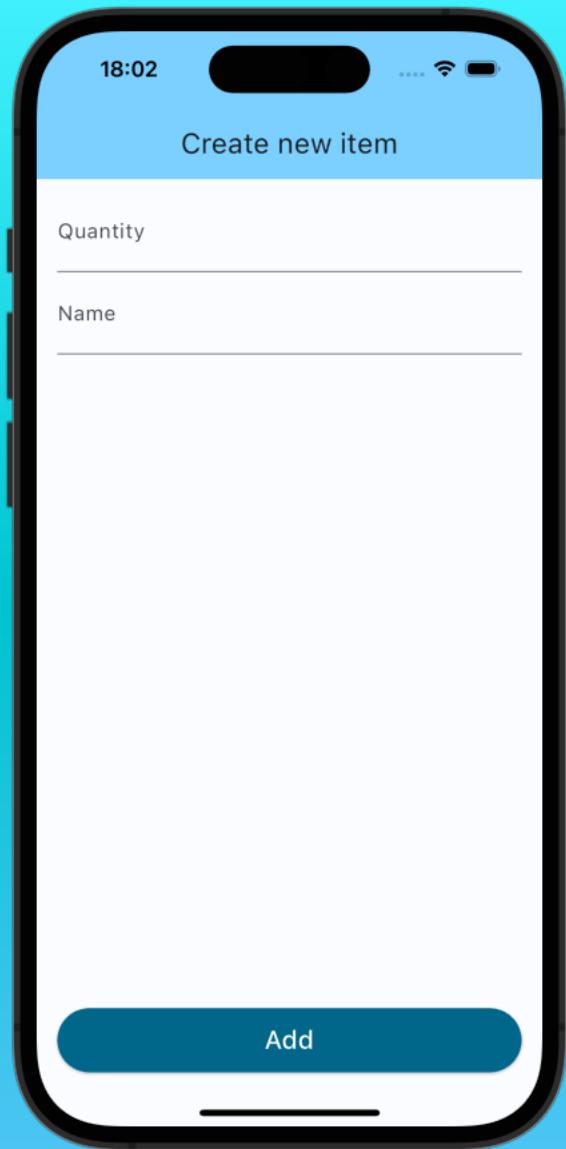
- Rerenders the whole widget tree when the input changes
- Useful when the input is displayed somewhere else than the TextField itself

TextEditingController

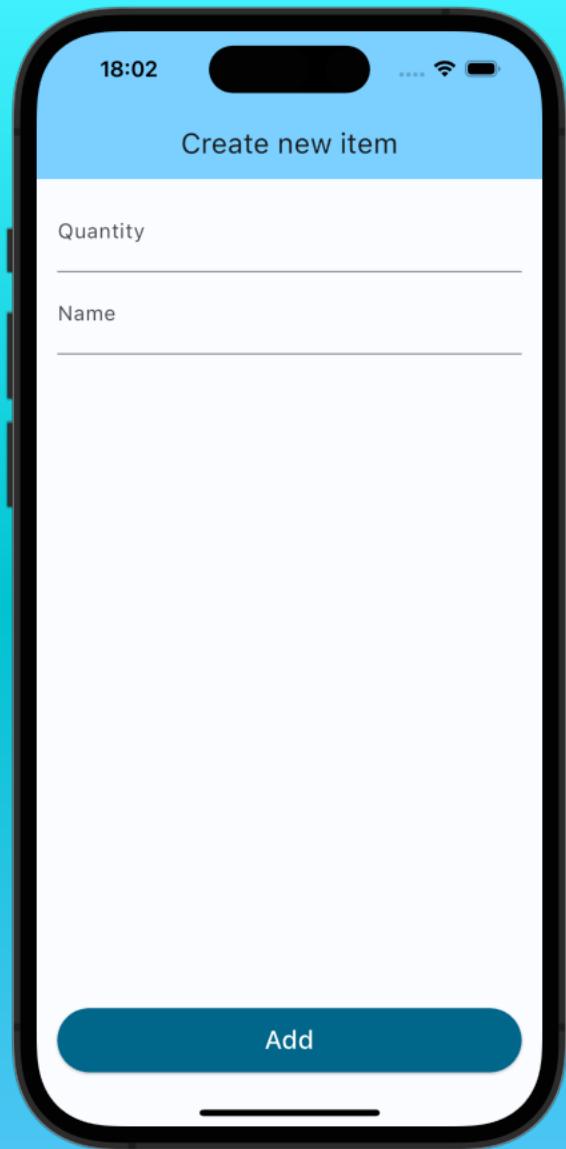
- Only rerenders the TextField
- Requires disposing and initializing



```
 class _CreateNewItemFormState extends  
State<_CreateNewItemForm> {  
    late final TextEditingController  
    _quantityController;  
    late final TextEditingController  
    _nameController;  
    ...  
}
```



```
class _CreateNewItemFormState extends  
State<_CreateNewItemForm> {  
    late final TextEditingController  
    _quantityController;  
    late final TextEditingController  
    _nameController;  
    ↳ tells the compiler that _nameController is initialized before  
} being used
```



```
class _CreateNewItemFormState extends State<_CreateNewItemForm> {
    late final TextEditingController
    _quantityController;
    late final TextEditingController
    _nameController;

    @override
    void initState() {
        super.initState();
        _quantityController =
    TextEditingController();
        _nameController = TextEditingController();
    }

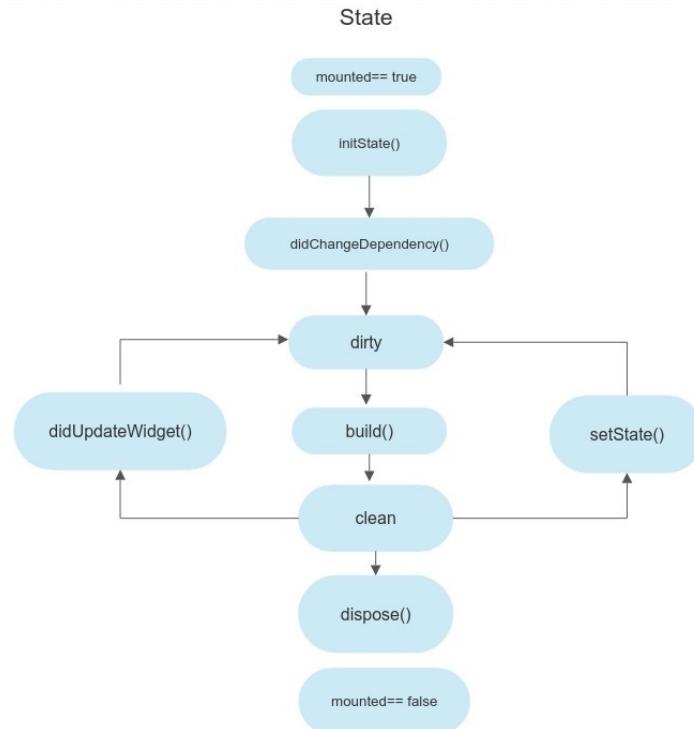
    ...
}
```

Lifecycle methods of a StatefulWidget

Shopping List App

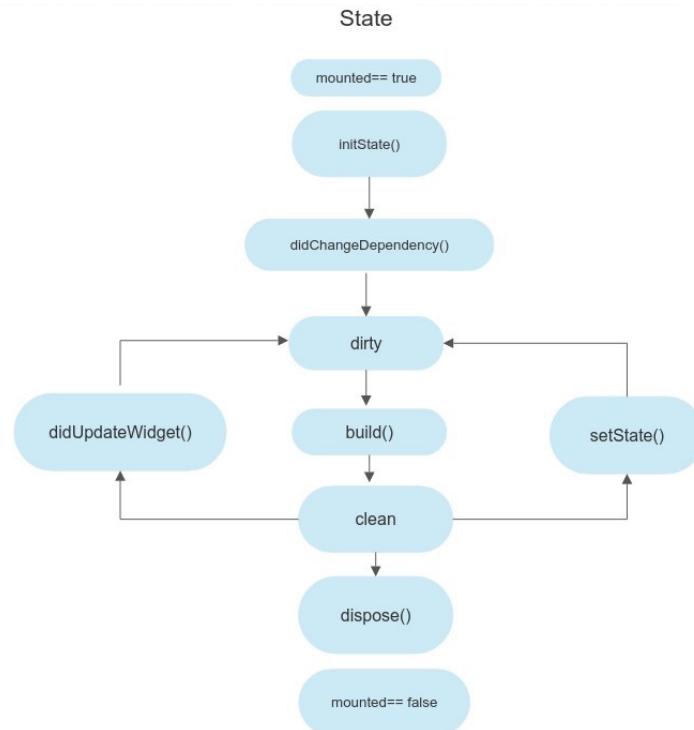
Lifecycle methods of a StatefulWidget (processed in order)

1. `createState()`
2. `mounted = true`
3. `initState()`
4. `didChangeDependencies()`
5. `build()`
6. `didUpdateWidget()`
7. `setState()`
8. `deactivate()`
9. `dispose()`
10. `mounted = false`



Lifecycle methods of a StatefulWidget (processed in order)

1. `createState()`
2. `mounted = true`
3. **`initState()`**
4. `didChangeDependencies()`
5. **`build()`**
6. `didUpdateWidget()`
7. **`setState()`**
8. `deactivate()`
9. **`dispose()`**
10. `mounted = false`

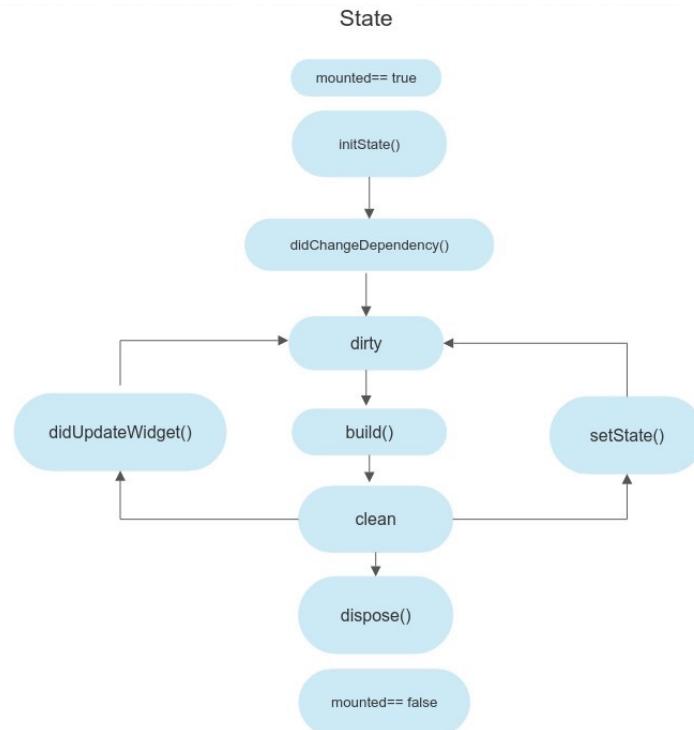


initState()

- Is the first method called after the creation of the widget
- Only called once!
- Must call `super.initState()` before doing anything else
- Useful for:
 - Initializing data that relies on the BuildContext
 - Initializing properties that rely on properties from the widget itself
 - Subscribing to any object that could change the data on the widget

Lifecycle methods of a StatefulWidget

1. `createState()`
2. `mounted = true`
3. `initState()`
4. `didChangeDependencies()`
5. **`build()`**
6. `didUpdateWidget()`
7. **`setState()`**
8. `deactivate()`
9. **`dispose()`**
10. `mounted = false`

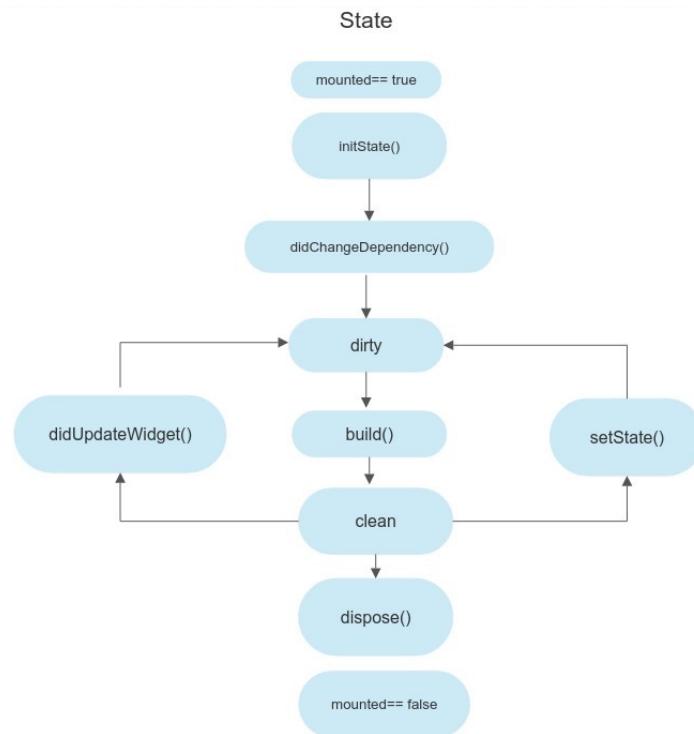


build()

- Called on every render => Called very often!
- Must return a widget

Lifecycle methods of a StatefulWidget

1. `createState()`
2. `mounted = true`
3. `initState()`
4. `didChangeDependencies()`
5. `build()`
6. `didUpdateWidget()`
7. `setState()`
8. `deactivate()`
9. `dispose()`
10. `mounted = false`

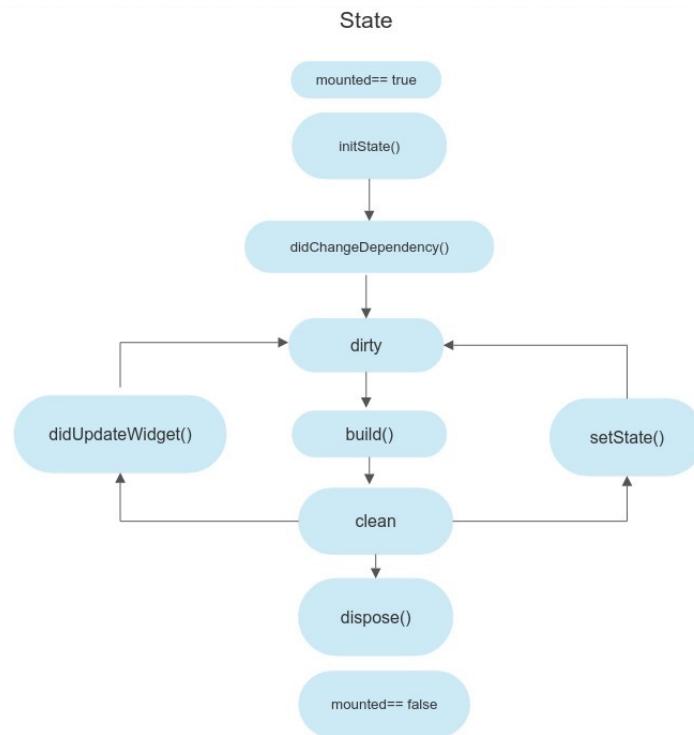


dispose()

- Called when the state is permanently removed from the tree
- Needs to call `super.dispose()` after everything else
- Useful for:
 - Unsubscribing
 - Cancelling animations
 - Calling dispose methods of state members (e.g. `TextEditingController`)

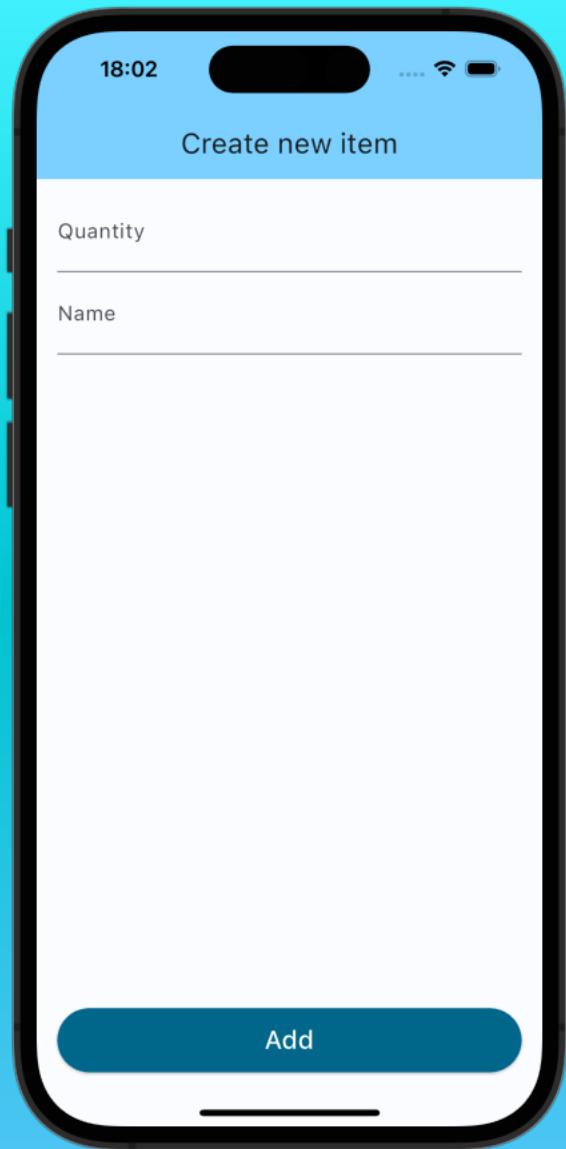
Lifecycle methods of a StatefulWidget

1. `createState()`
 2. `mounted = true`
 3. `initState()`
 4. `didChangeDependencies()`
 5. `build()`
 6. `didUpdateWidget()`
 7. `setState()`
 8. `deactivate()`
 9. `dispose()`
 10. `mounted = false`



Handling the lifecycle of a `TextEditingController`

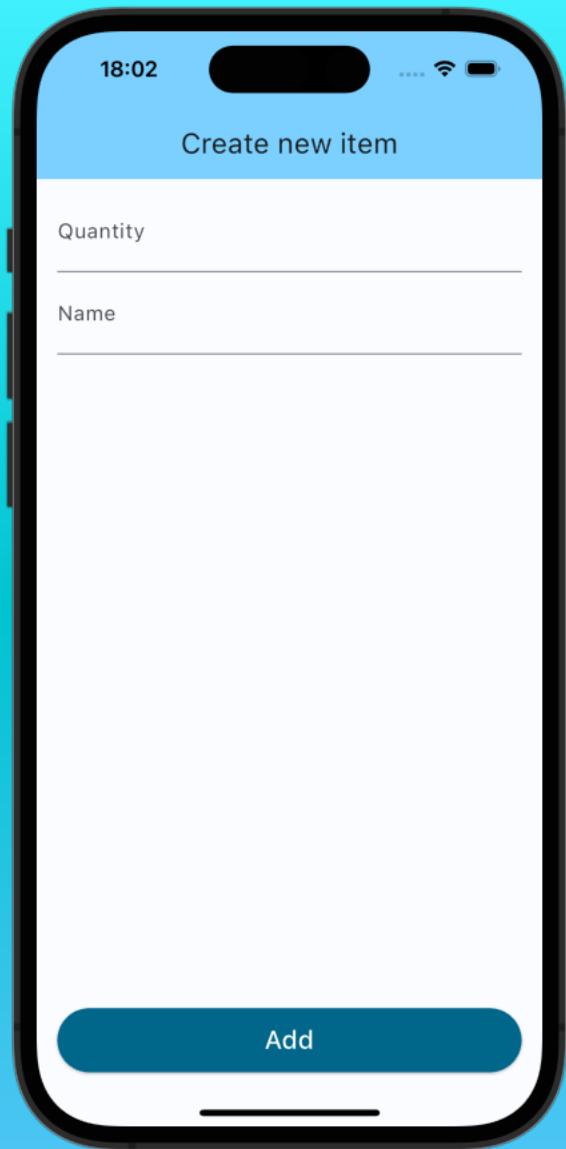
Shopping List App



```
class _CreateNewItemFormState extends State<_CreateNewItemForm> {
    late final TextEditingController
    _quantityController;
    late final TextEditingController
    _nameController;

    @override
    void initState() {
        super.initState();
        _quantityController =
    TextEditingController();
        _nameController = TextEditingController();
    }

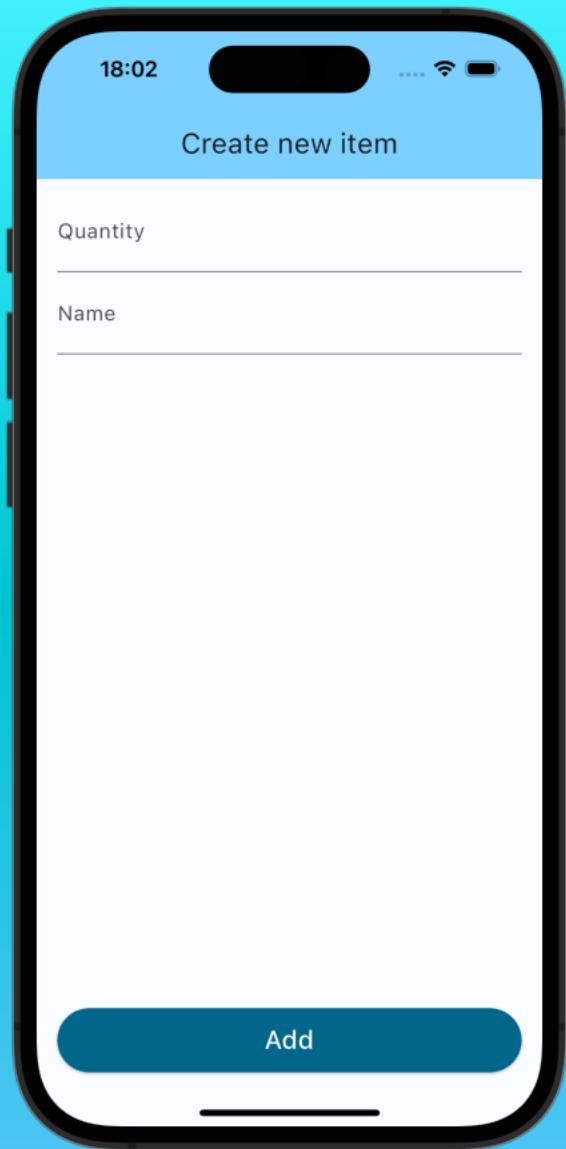
    ...
}
```



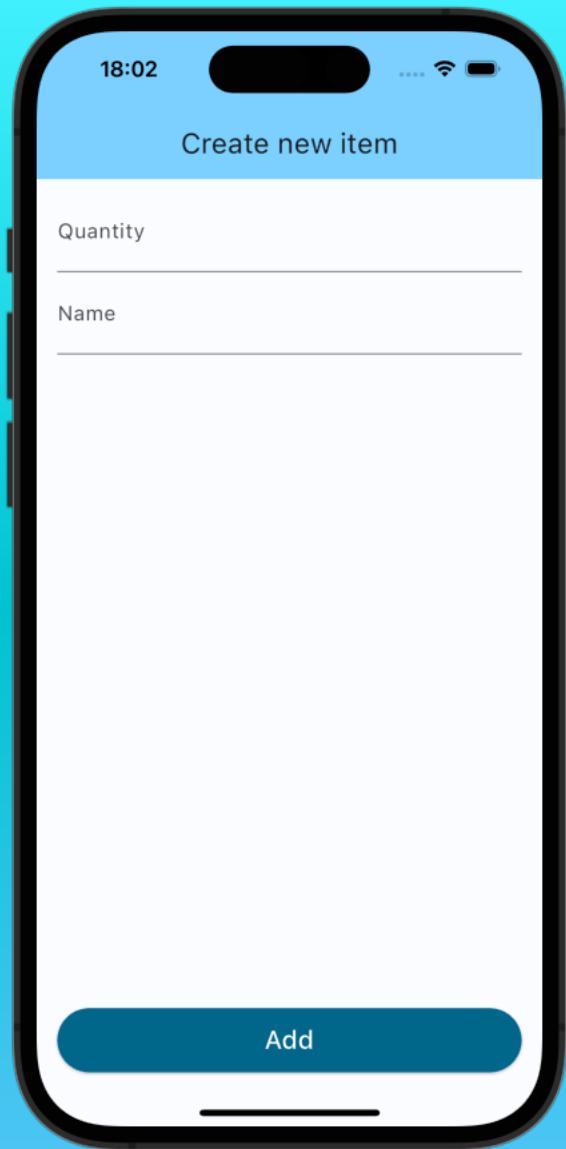
```
class _CreateNewItemFormState extends State<_CreateNewItemForm> {
    late final TextEditingController _quantityController;
    late final TextEditingController _nameController;

    @override
    void initState() {
        super.initState();
        _quantityController = TextEditingController();
        _nameController = TextEditingController();
    }

    @override
    void dispose() {
        _quantityController.dispose();
        _nameController.dispose();
        super.dispose();
    }
}
```



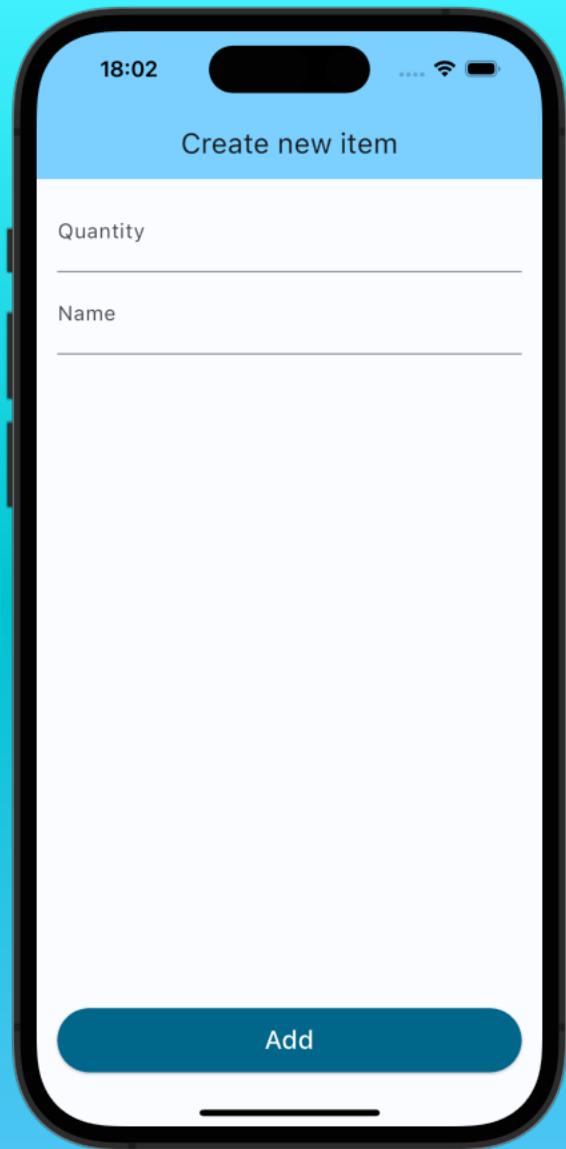
```
[  
  TextFormField(  
    decoration: InputDecoration(  
      labelText: 'Quantity',  
    ),  
  ),  
  TextFormField(  
    decoration: InputDecoration(  
      labelText: 'Name',  
    ),  
  ),  
]
```



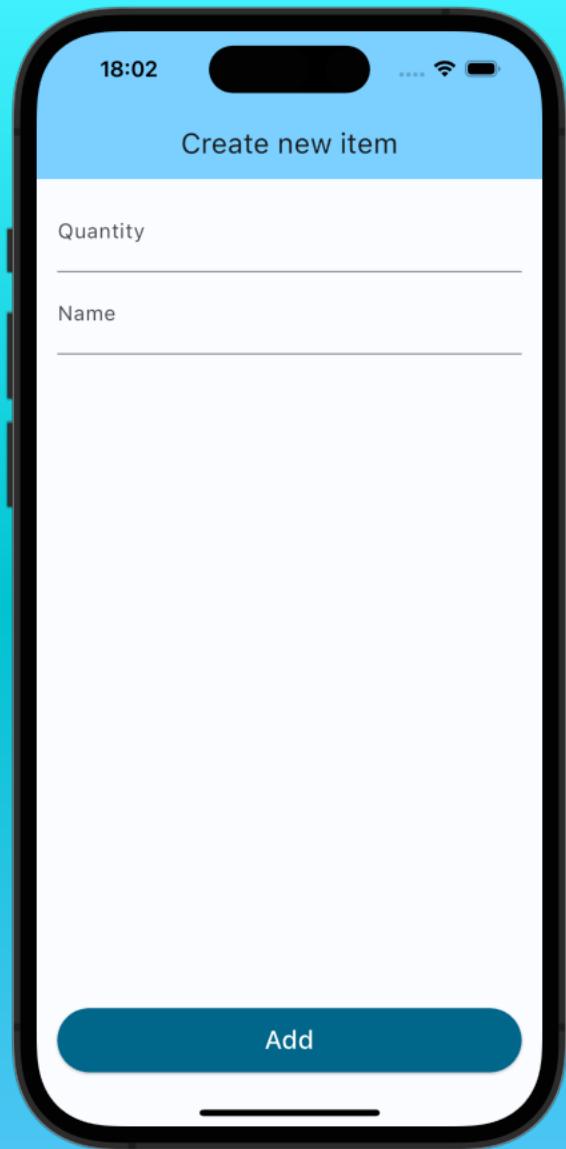
```
[  
  TextFormField(  
    controller: _quantityController,  
    decoration: InputDecoration(  
      labelText: 'Quantity',  
    ),  
  ),  
  TextFormField(  
    controller: _nameController,  
    decoration: InputDecoration(  
      labelText: 'Name',  
    ),  
  ),  
]
```

Adding Validation

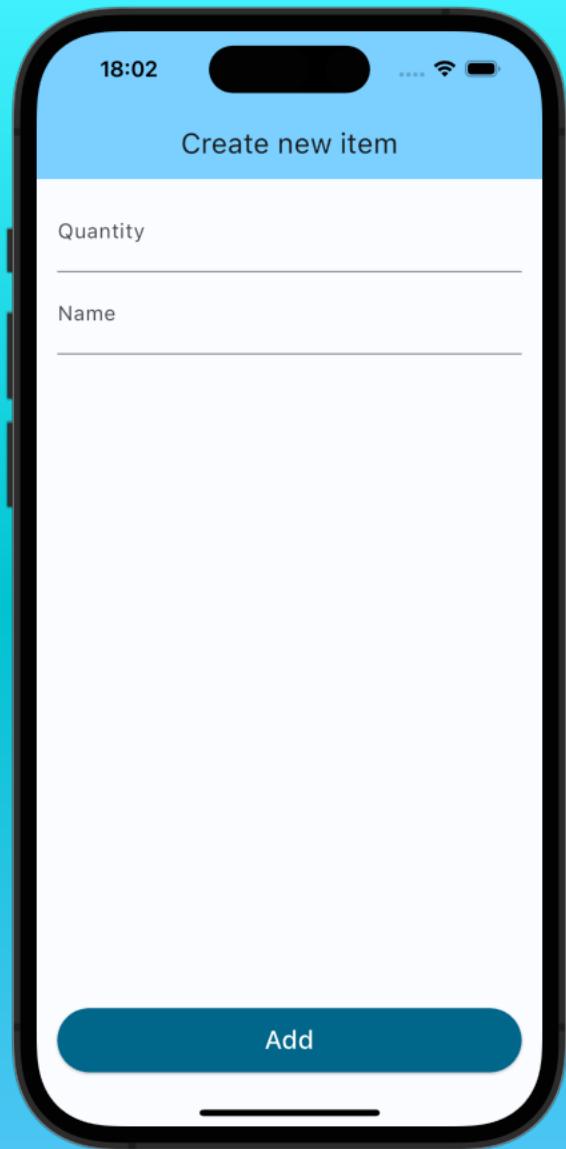
Shopping List App



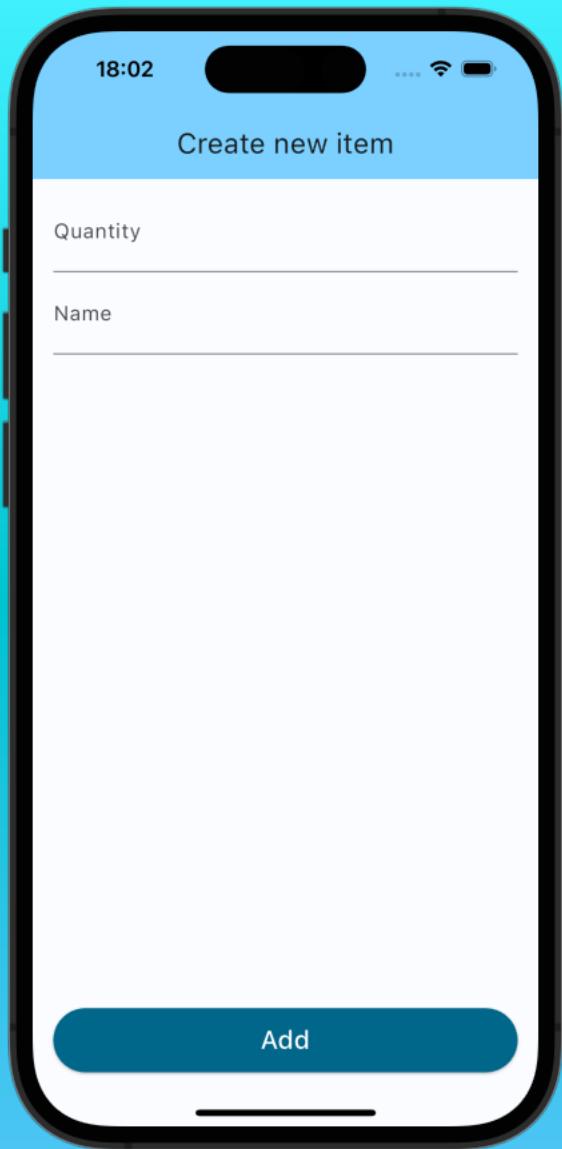
```
[  
  TextFormField(  
    controller: _quantityController,  
    decoration: InputDecoration(  
      labelText: 'Quantity',  
    ),  
  ),  
  TextFormField(  
    controller: _nameController,  
    decoration: InputDecoration(  
      labelText: 'Name',  
    ),  
  ),  
]
```



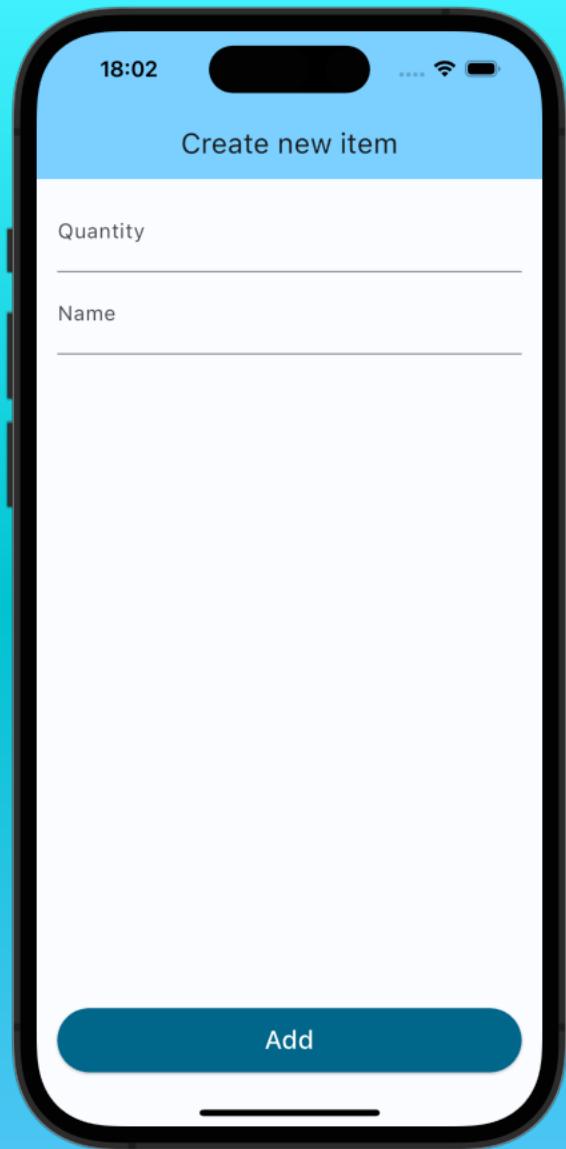
```
[  
  TextFormField(  
    controller: _quantityController,  
    keyboardType: TextInputType.numberWithOptions(  
      decimal: false,  
      signed: false,  
    ),  
    decoration: InputDecoration(  
      labelText: 'Quantity',  
    ),  
  ),  
  TextFormField(  
    validator: (value) {  
      if (value == null || value.isEmpty) {  
        return 'Please enter a name';  
      }  
      return null;  
    },  
    controller: _nameController,  
    decoration: InputDecoration(  
      labelText: 'Name',  
    ),  
  ),  
>]
```



```
final _formKey = GlobalKey<FormState>();  
{@override  
Widget build(BuildContext context) {  
    return SafeArea(  
        child: Padding(  
            padding: ...,  
            child: Form(  
                key: _formKey,  
                child: ...  
            ),  
        ),  
    );  
}}
```



```
ElevatedButton(  
    style: ElevatedButton.styleFrom(  
        backgroundColor:  
Theme.of(context).colorScheme.primary,  
        foregroundColor:  
Theme.of(context).colorScheme.onPrimary,  
        minimumSize: const Size.fromHeight(50.0),  
    ),  
    onPressed: () {},  
    child: const Text(  
        'Add',  
        style: TextStyle(fontSize: 20),  
    ),  
,
```

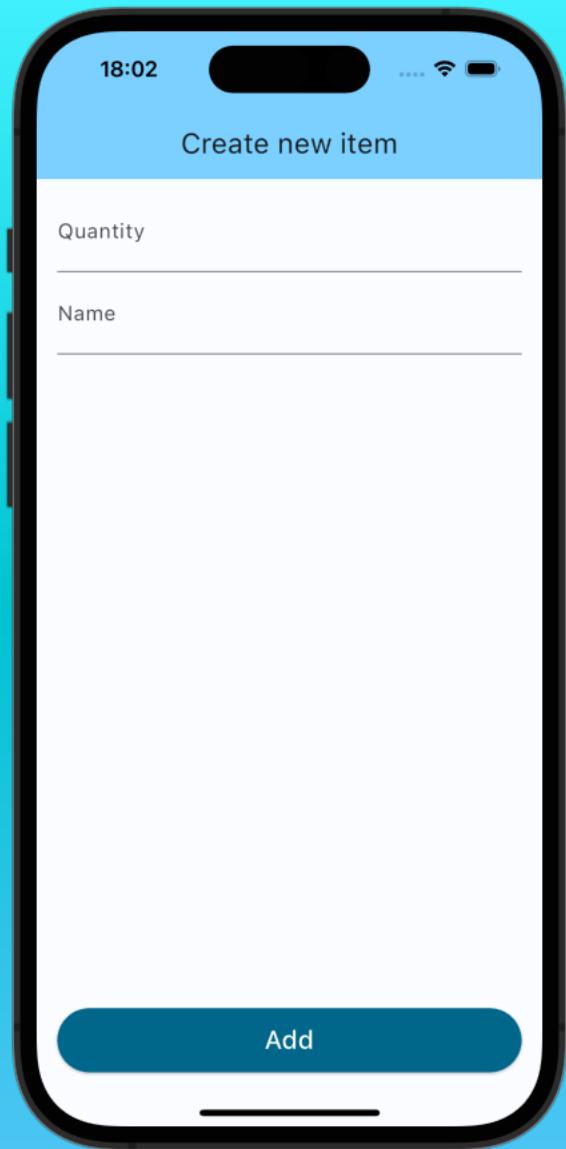


```
ElevatedButton(  
    style: ...,  
    onPressed: () {  
        if (!_formKey.currentState!.validate()) {  
            return;  
        }  
        ,  
        child: ...,  
    ),
```

A dark gray rectangular box contains a snippet of Dart code. The code defines an ElevatedButton widget. It includes styling parameters like 'style' and 'onPressed'. The 'onPressed' handler contains a conditional statement that checks the validation state of a form key. If validation fails, it returns early. Otherwise, it proceeds with the button's child content.

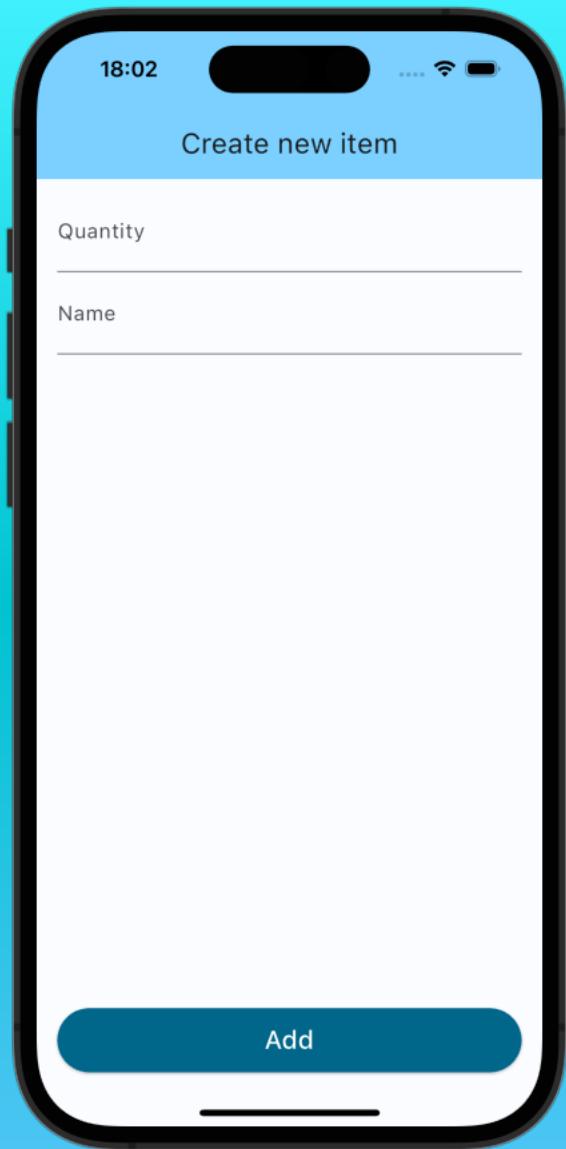
Extracting Input

Shopping List App



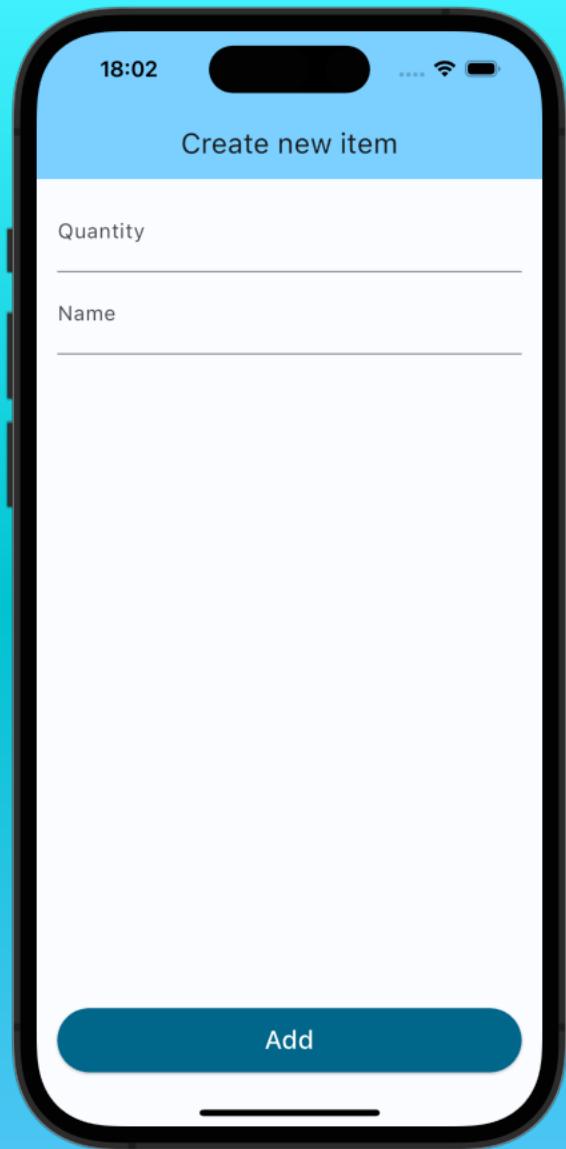
```
ElevatedButton(  
    style: ...,  
    onPressed: () {  
        if (!_formKey.currentState!.validate()) {  
            return;  
        }  
        ,  
        child: ...,  
    ),
```

A dark gray rectangular box contains a snippet of Dart code. The code defines an ElevatedButton widget. It includes styling parameters like 'style: ...' and an 'onPressed' event handler. Inside the event handler, there is a check using the '_formKey.currentState!.validate()' method. If the validation fails, the function returns early. Otherwise, it continues with a comma-separated list of children, which includes another comma and a 'child: ...' parameter.

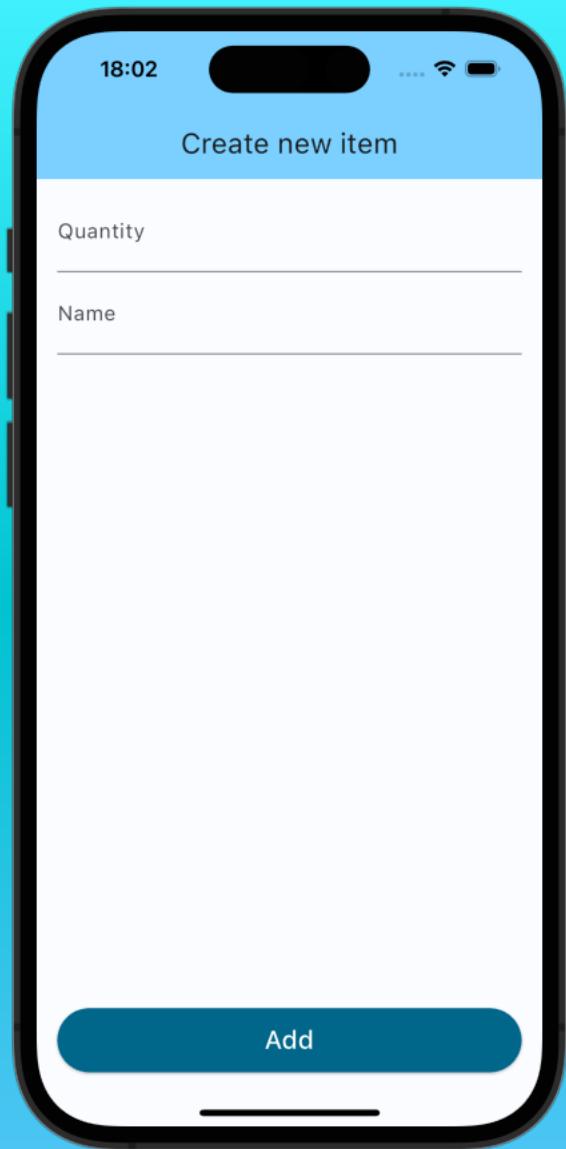


```
ElevatedButton(  
    style: ...,  
    onPressed: () {  
        if (!_formKey.currentState!.validate()) {  
            return;  
        }  
        final quantity =  
int.tryParse(_quantityController.text) ?? 1;  
        final name = _nameController.text;  
    },  
    child: ...,  
) ,
```

A dark gray rectangular box contains the code for a Flutter ElevatedButton. The code uses Dart syntax to define the button's properties. It includes a validation check using the '_formKey.currentState!.validate()' method, and it attempts to parse the 'Quantity' text into an integer using 'int.tryParse()'. If successful, it retrieves the 'Name' from the '_nameController'. The code ends with a closing parenthesis for the ElevatedButton and a comma, likely indicating it's part of a larger list or map.



```
ElevatedButton(  
    style: ...,  
    onPressed: () {  
        if (!_formKey.currentState!.validate()) {  
            return;  
        }  
        final quantity =           Null operator ↑  
        int.tryParse(_quantityController.text) ?? 1;  
        final name = _nameController.text;  
    },  
    child: ...,  
) ,
```



```
ElevatedButton(  
    style: ...,  
    onPressed: () {  
        if (!_formKey.currentState!.validate()) {  
            return;  
        }  
        final quantity =  
int.tryParse(_quantityController.text) ?? 1;  
        final name = _nameController.text;  
        widget.shoppingList.addItem(  
            name: name,  
            quantity: quantity,  
        );  
    },  
    child: ...,  
),
```

Navigation to the new Page

Shopping List App

Navigators

Navigator 1.0

- Imperative API
- Limited control over navigation stack
- Easy to use
- Sufficient for many usecases

Navigator 2.0

- Declarative API
- Full control
- Complex
- Mostly used via third party packages like auto_route or go_router

Navigators

Navigator 1.0

- Imperative API
- Limited control over navigation stack
- Easy to use
- Sufficient for many usecases

👉 We will use the simpler approach

Navigator 2.0

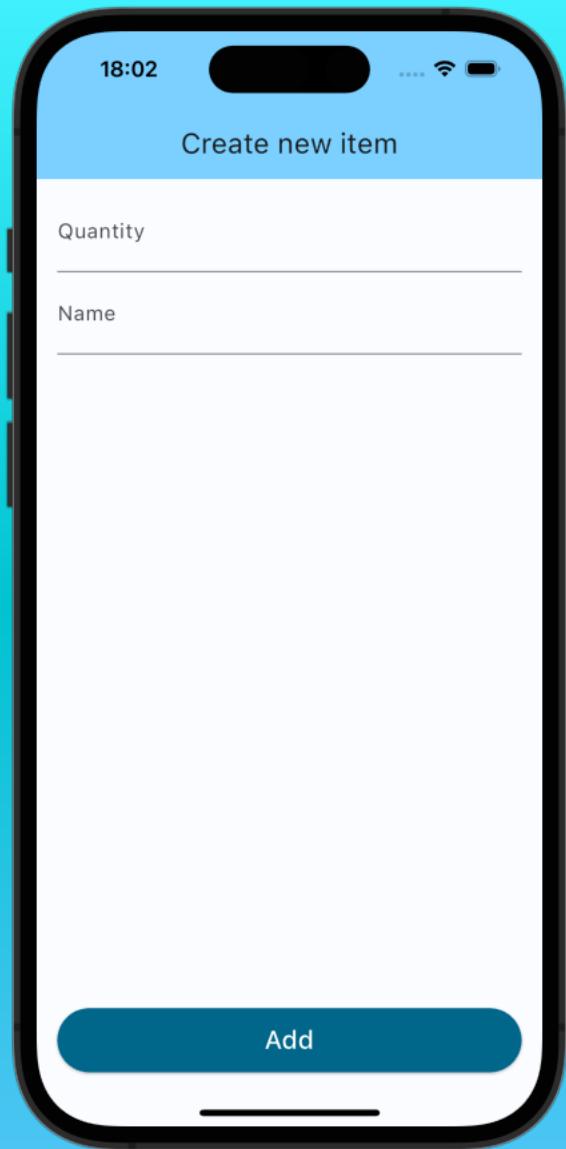
- Declarative API
- Full control
- Complex
- Mostly used via third party packages like auto_route or go_router

Pushing new pages

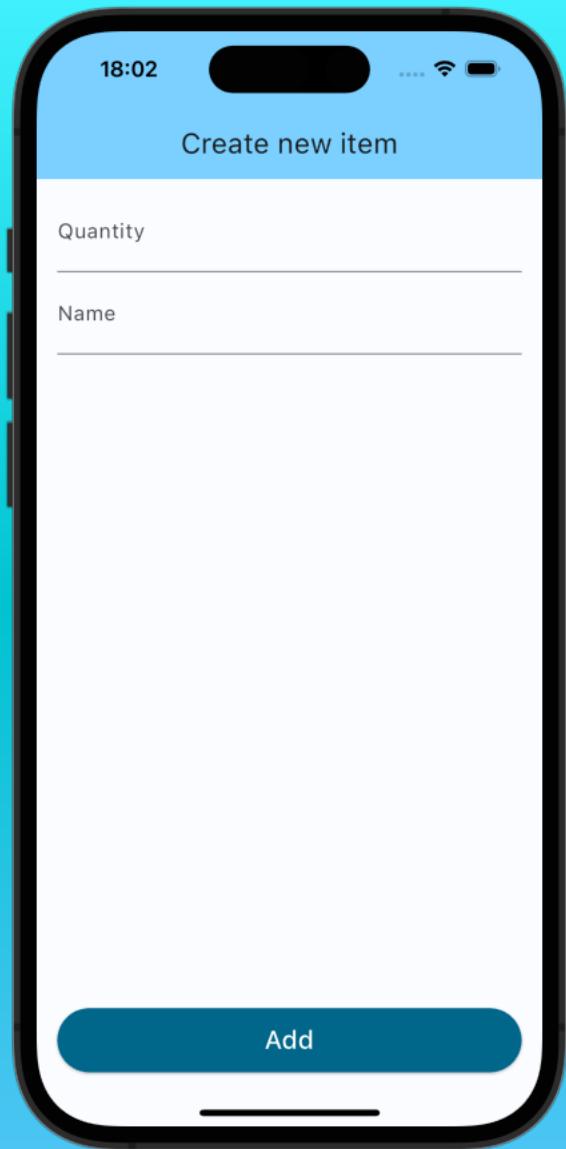
```
● ● ●  
  
// Within the `FirstRoute` widget  
 onPressed: () {  
   Navigator.push(  
     context,  
     MaterialPageRoute(builder: (context) => const SecondRoute()),  
   );  
 }  
  
}
```

Popping pages

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

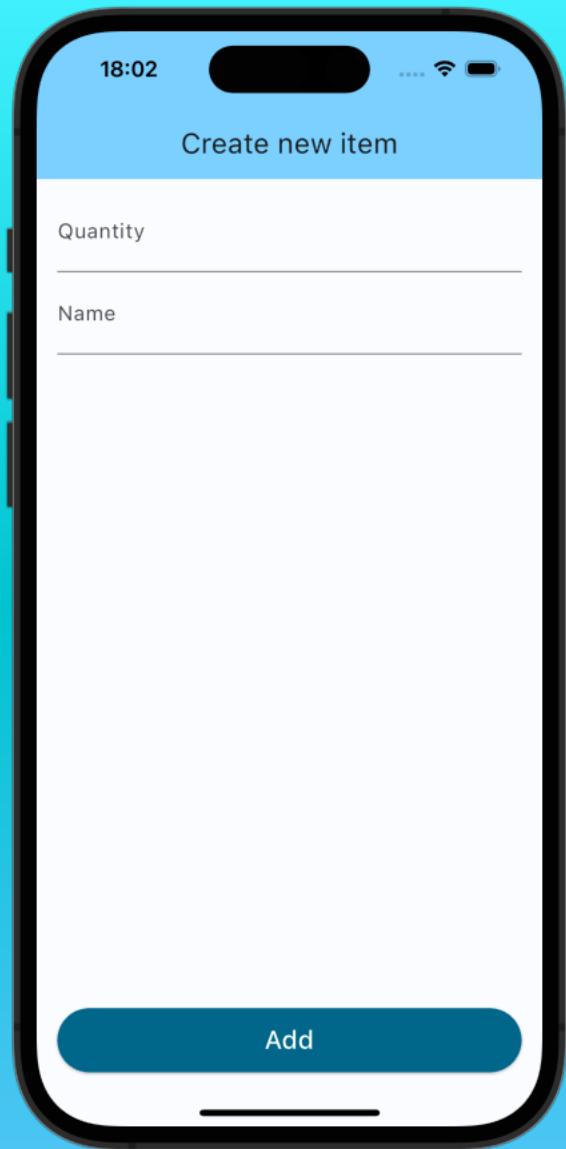


```
ElevatedButton(
    style: ...,
    onPressed: () {
        if (!_formKey.currentState!.validate()) {
            return;
        }
        final quantity =
int.tryParse(_quantityController.text) ?? 1;
        final name = _nameController.text;
        widget.shoppingList.addItem(
            name: name,
            quantity: quantity,
        );
    },
    child: ...,
),
```

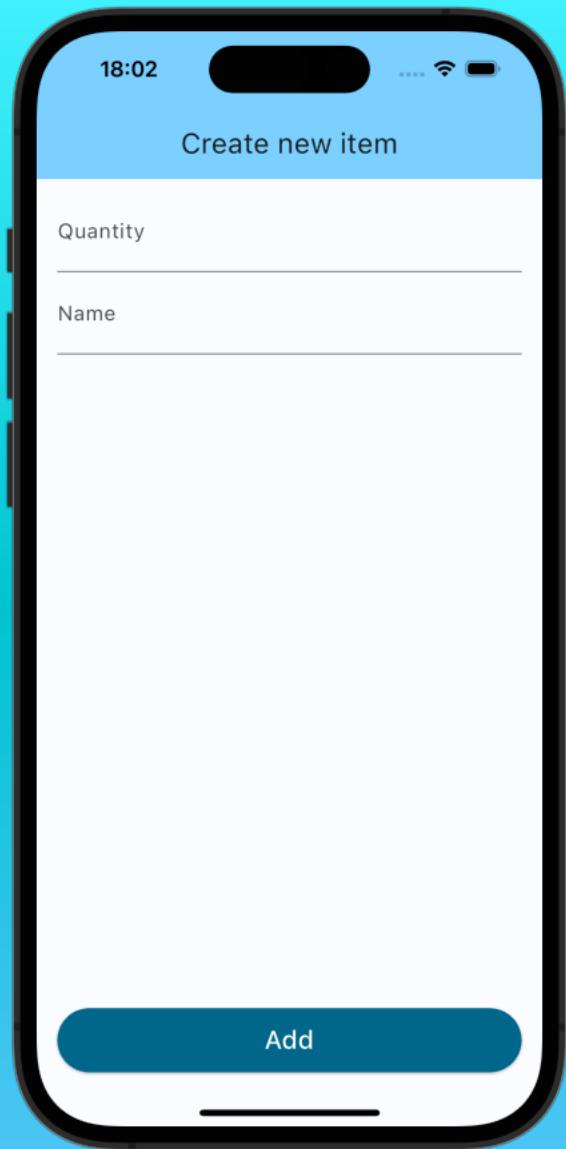


```
ElevatedButton(  
    style: ...,  
    onPressed: () {  
        if (!_formKey.currentState!.validate()) {  
            return;  
        }  
        final quantity =  
            int.tryParse(_quantityController.text) ?? 1;  
        final name = _nameController.text;  
        widget.shoppingList.addItem(  
            name: name,  
            quantity: quantity,  
        );  
        Navigator.pop(context);  
    },  
    child: ...,  
),
```

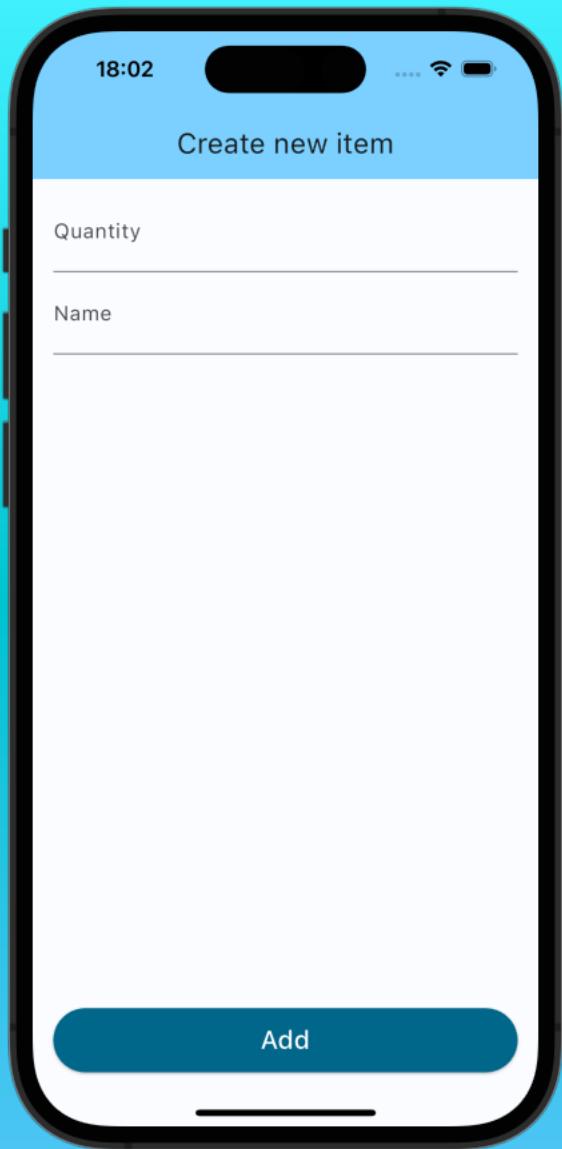
A dark gray rectangular panel is positioned to the right of the phone. It features the Flutter logo icon (a stylized blue "F") on the left. To its right is a block of Dart code. The code defines an ElevatedButton. It includes logic to validate form state, parse the quantity from a text controller, and add a new item to a shopping list. Finally, it uses Navigator.pop to close the current screen.



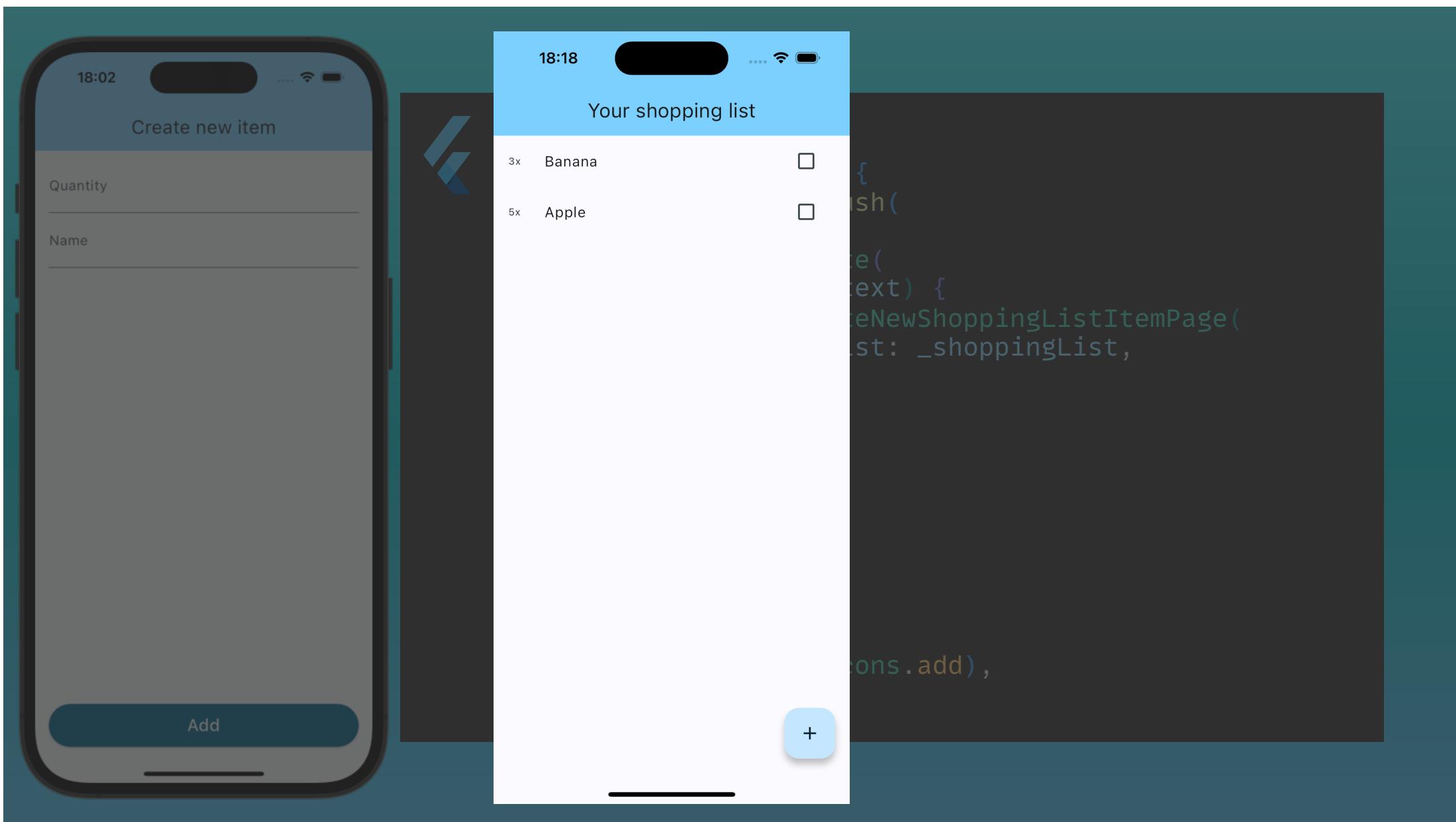
```
FloatingActionButton(  
    onPressed: () {  
        setState(() {  
            _shoppingList.addItem(name: 'New item',  
            quantity: 1);  
        });  
    },  
    child: const Icon(Icons.add),  
)
```

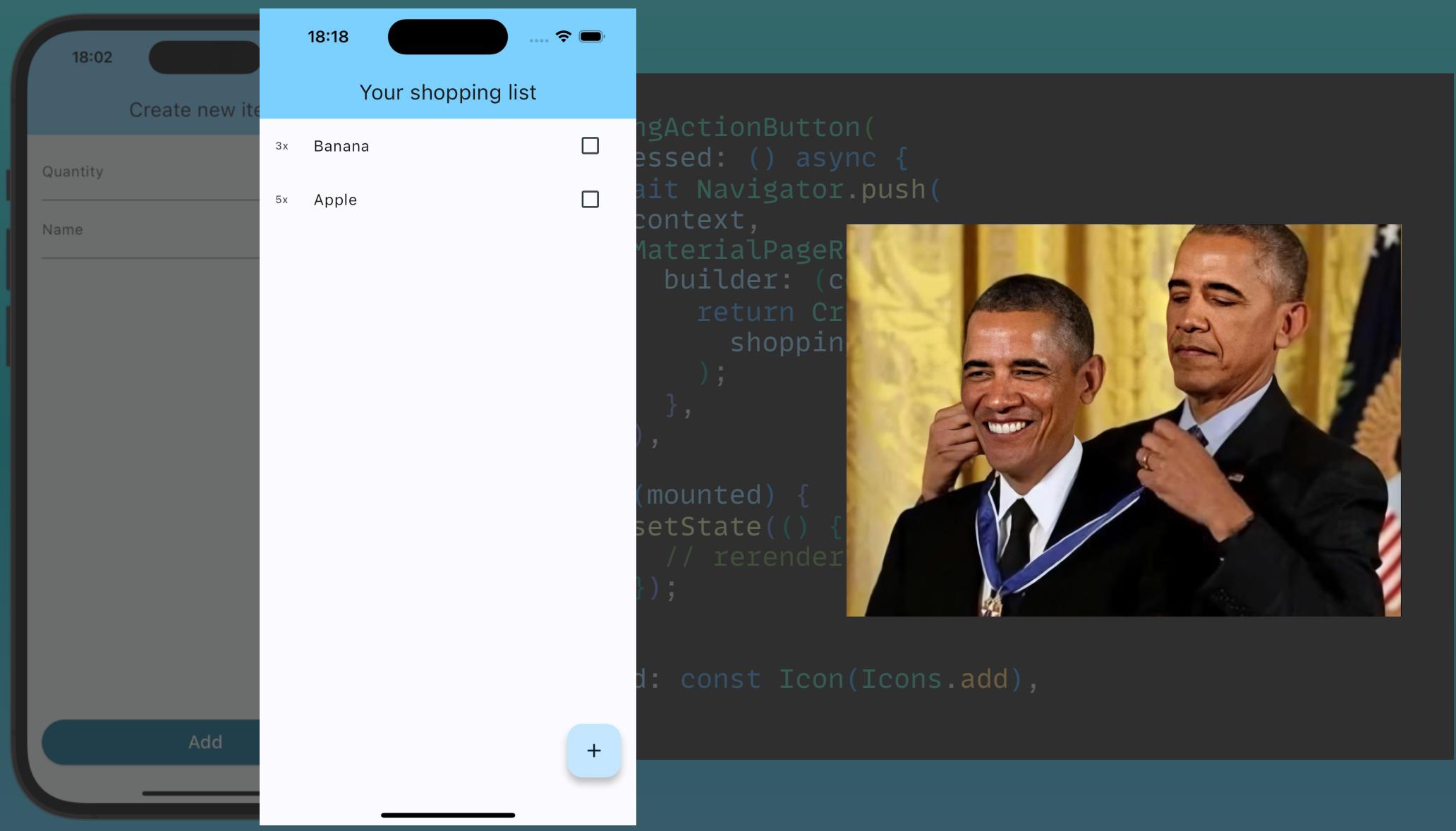


```
FloatingActionButton(  
    onPressed: () async {  
        await Navigator.push(  
            context,  
            MaterialPageRoute(  
                builder: (context) {  
                    return CreateNewShoppingListWidgetItemPage(  
                        shoppingList: _shoppingList,  
                    );  
                },  
            ),  
        );  
    },  
    child: const Icon(Icons.add),  
)
```



```
FloatingActionButton(  
    onPressed: () async {  
        await Navigator.push(  
            context,  
            MaterialPageRoute(  
                builder: (context) {  
                    return CreateNewItemPage(  
                        shoppingList: _shoppingList,  
                    );  
                },  
            ),  
        );  
        if(mounted) {  ↩ check if the widget is still mounted  
            setState(() {  after an asynchronous operation  
                // rerender  
            });  
        }  
    },  
    child: const Icon(Icons.add),  
)
```





Summary

- Widgets are used to configure other widgets
- There is a widget for everything
- Rerender using setState
- Navigate between pages using the Navigator API
- Use lifecycle methods to handle lifetime of created objects



Practice makes perfect

Build your own app

Technical Requirements

- Flutter installation

Technical Requirements

- Flutter installation



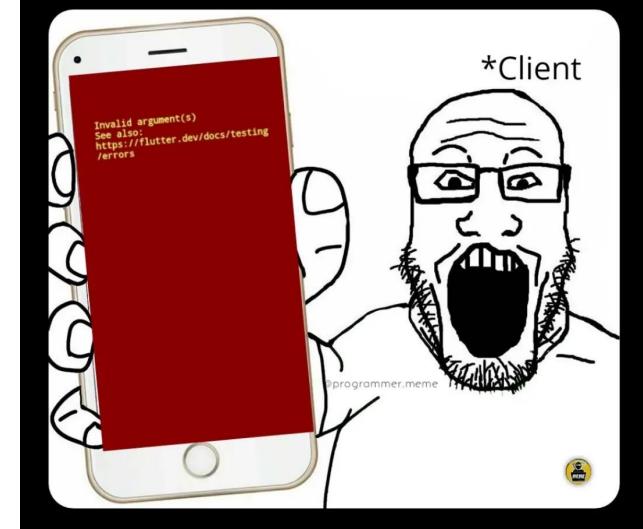
Technical Requirements

- Flutter installation
- A physical device or an emulator

Technical Requirements

- Flutter installation
- A physical device or an emulator
- A little patience

*Client when the UI
is not loaded in the
Flutter application



App Requirements

- Interactive elements that demonstrate stateful behavior (e.g. toggling the status of an item)

App Requirements

- Interactive elements that demonstrate stateful behavior (e.g. toggling the status of an item)
- Some kind of input handling (e.g. adding new items to a shopping list)

App Requirements

- Interactive elements that demonstrate stateful behavior (e.g. toggling the status of an item)
- Some kind of input handling (e.g. adding new items to a shopping list)
- At least one opportunity to navigate to a new screen

App Requirements

- Interactive elements that demonstrate stateful behavior (e.g. toggling the status of an item)
- Some kind of input handling (e.g. adding new items to a shopping list)
- At least one opportunity to navigate to a new screen
- Visual feedback (lookup Snackbar, Dialog, Animation)

App Ideas

- To-Do List
- Expense tracker
- Simple Quiz App
- Note-Taking App
- Personal Journal/Diary App
- Pomodoro Timer (Productivity timer that cycles between work intervals & breaks)

It's a competition





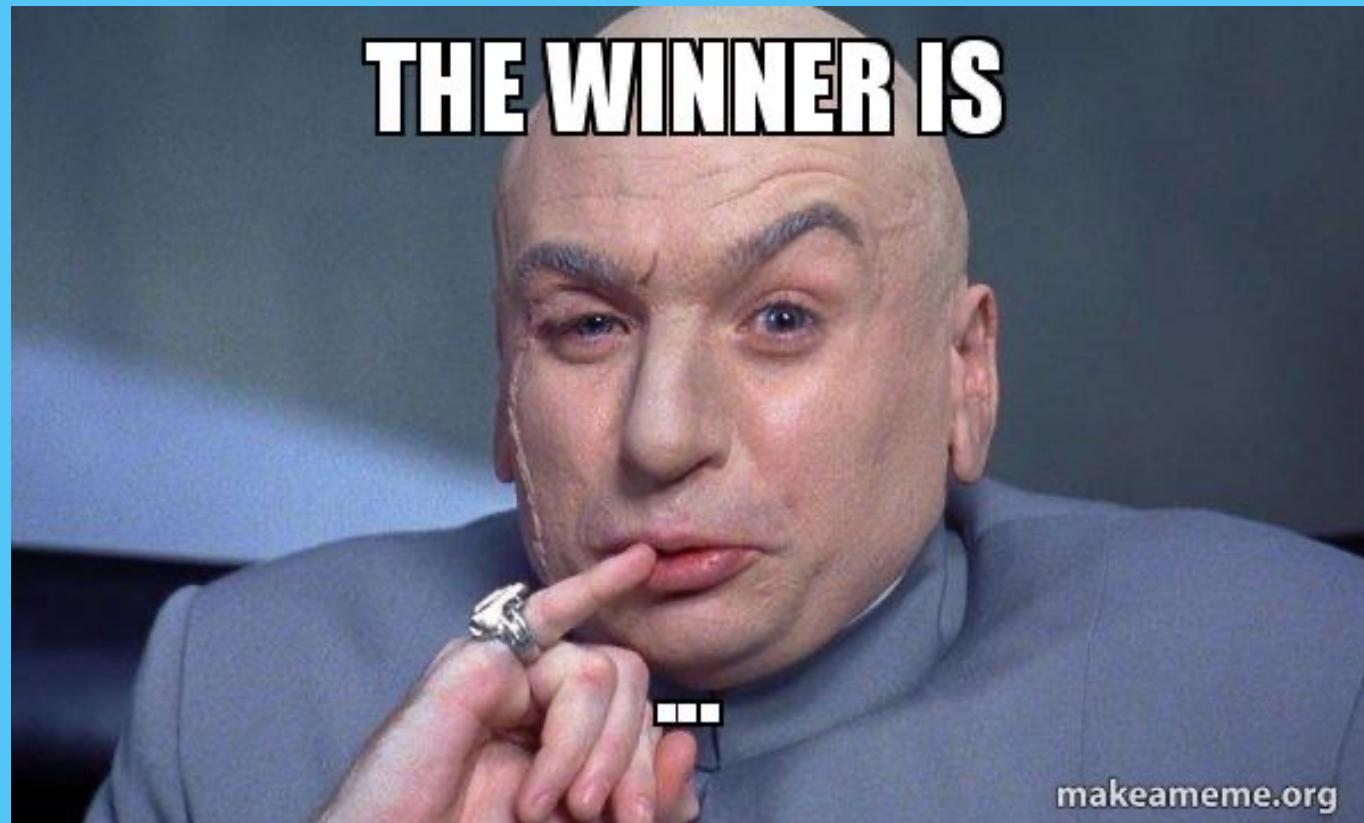
**Don't be afraid to ask
for help**



Apps & slides



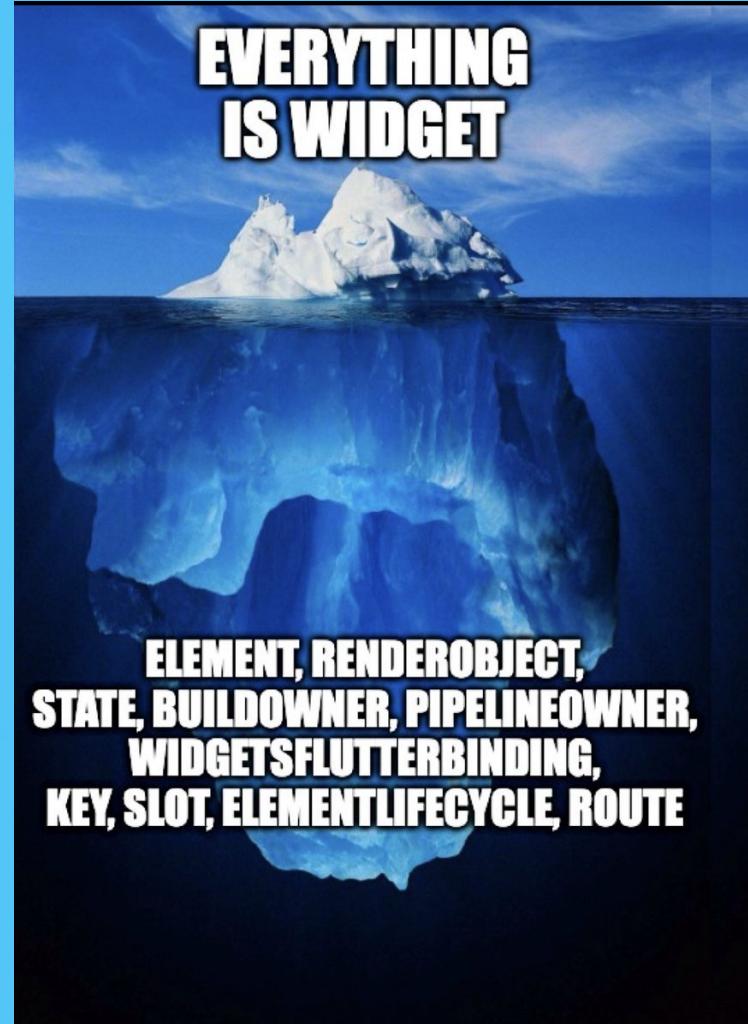
THE WINNER IS



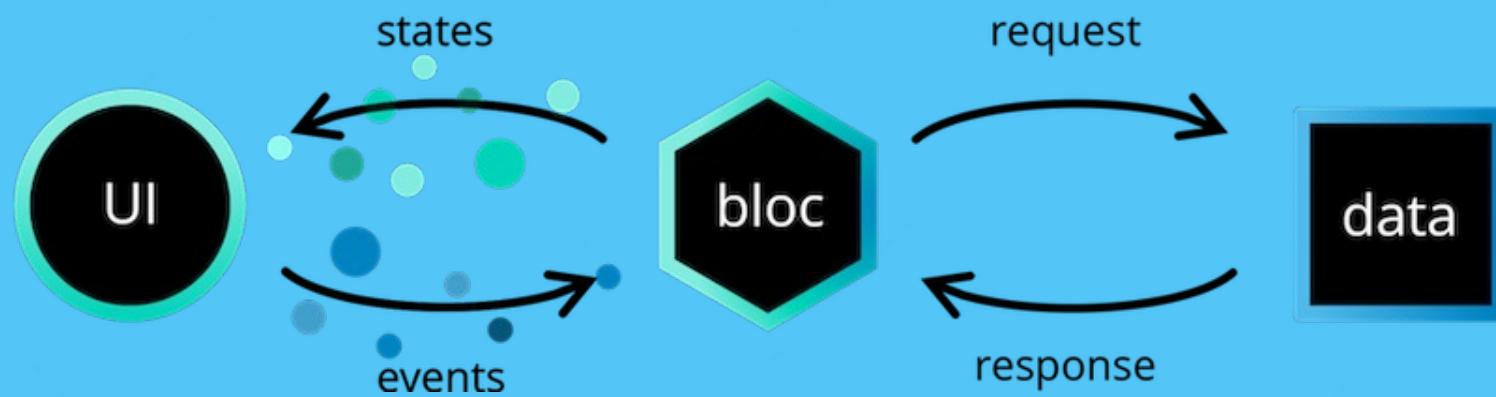


memegenerator.net

Outlook





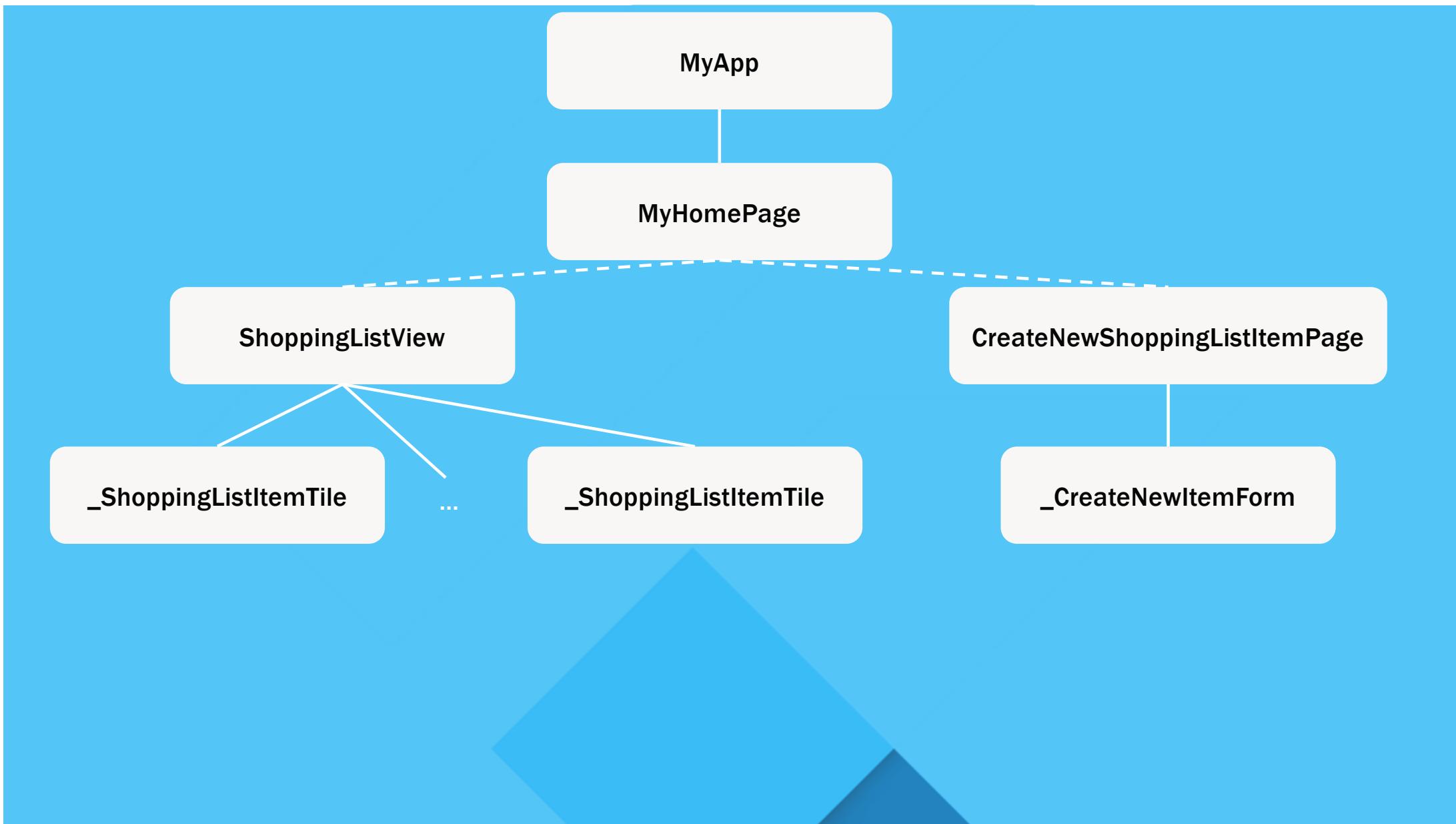


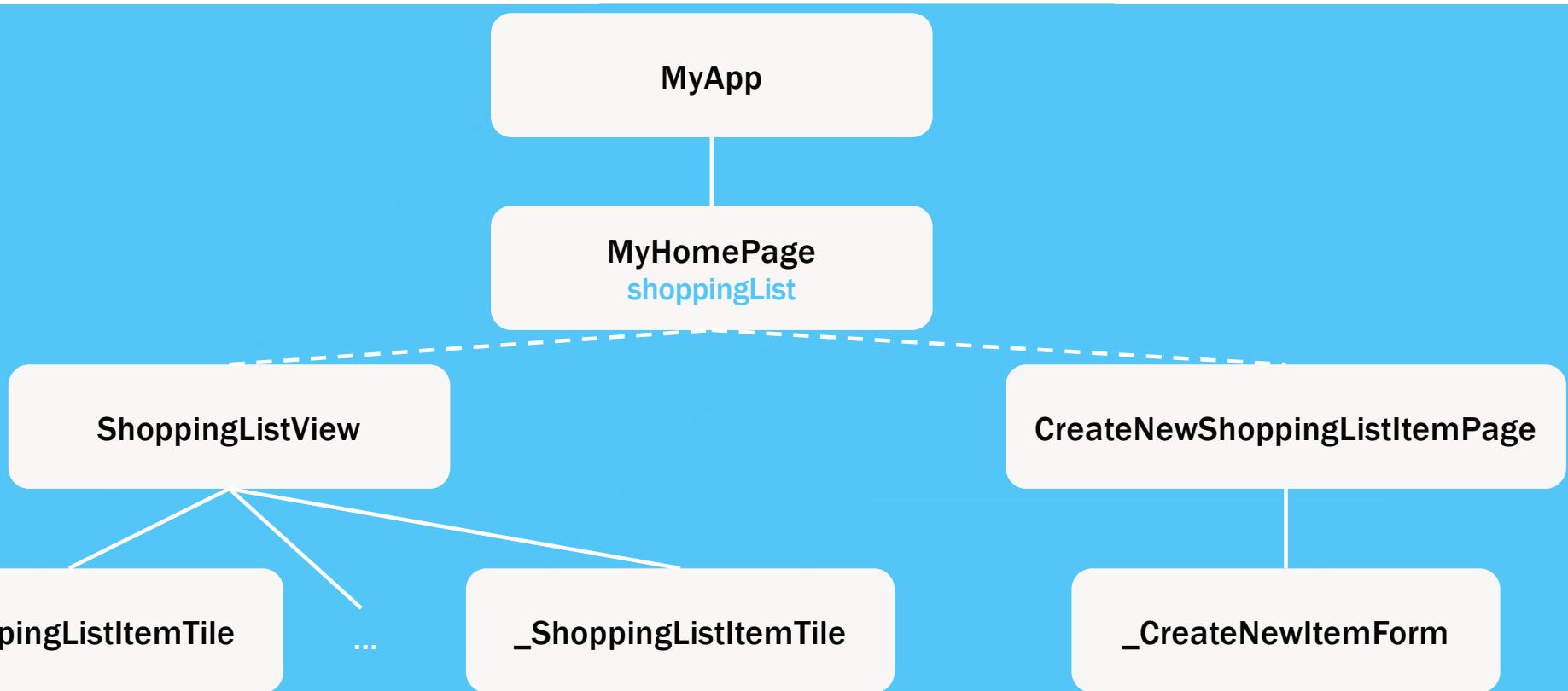
Things to lookup yourself

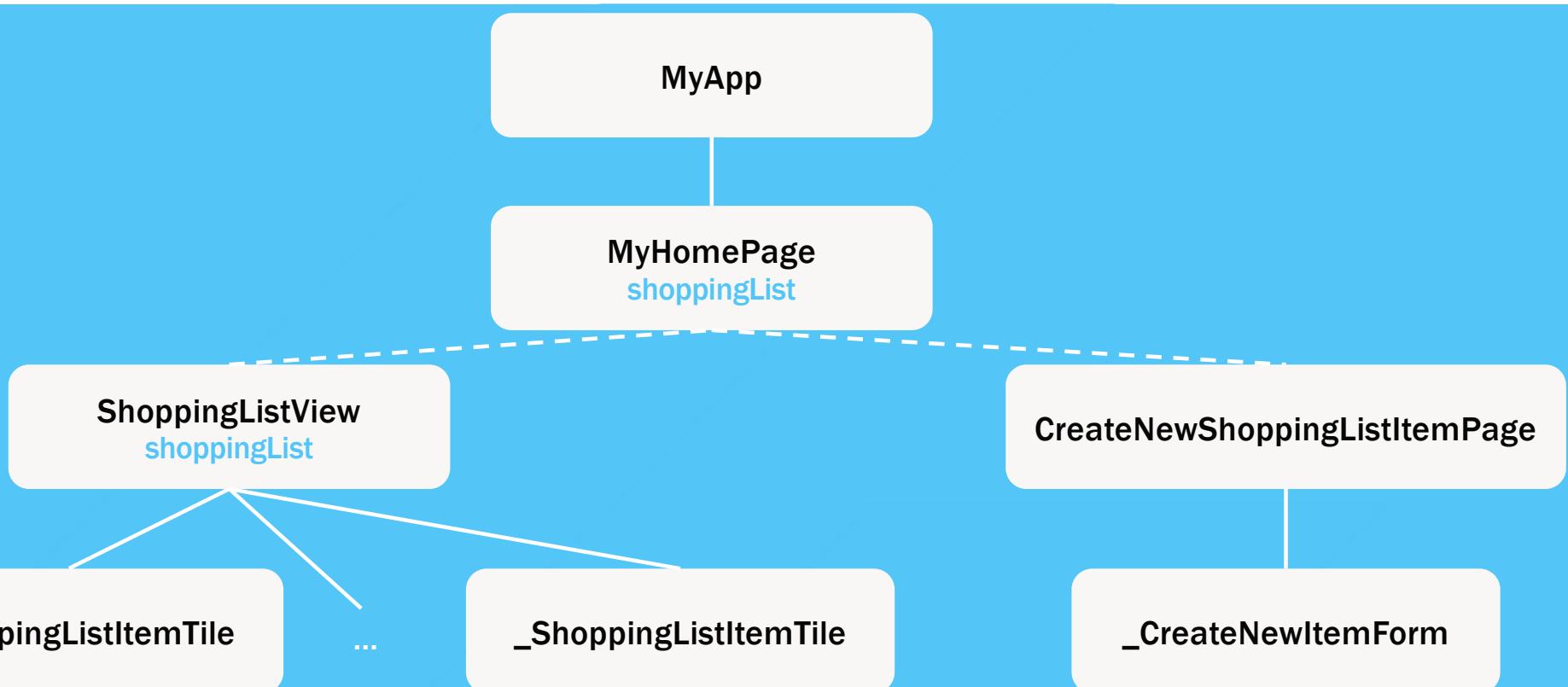
- InheritedWidget
- Element & RenderObject
- Bloc & Provider
- Asynchronous Dart => Call a REST-API from your Flutter app
- Freezed & JSON Serializable
- Widget/Unit/Integration Tests
- Local Storage

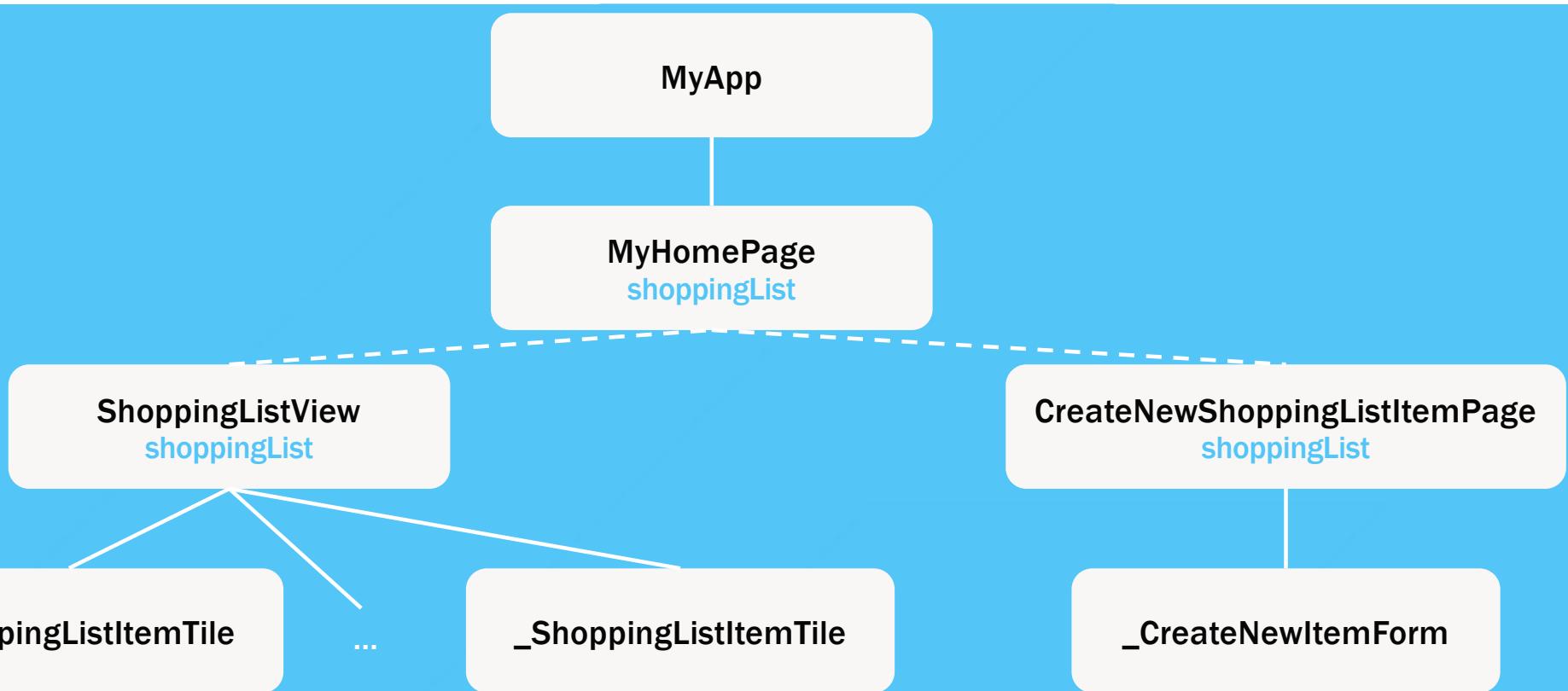
Advanced State Management

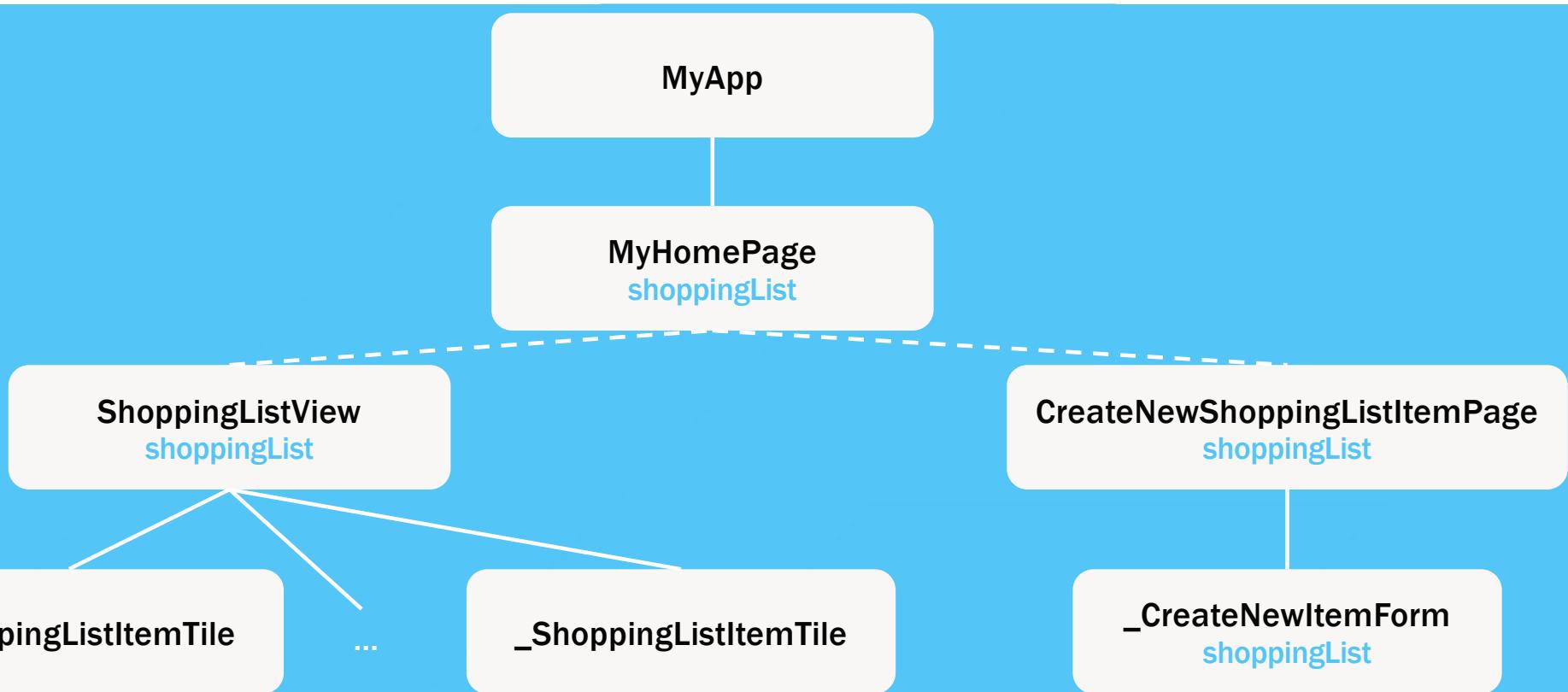
The problem

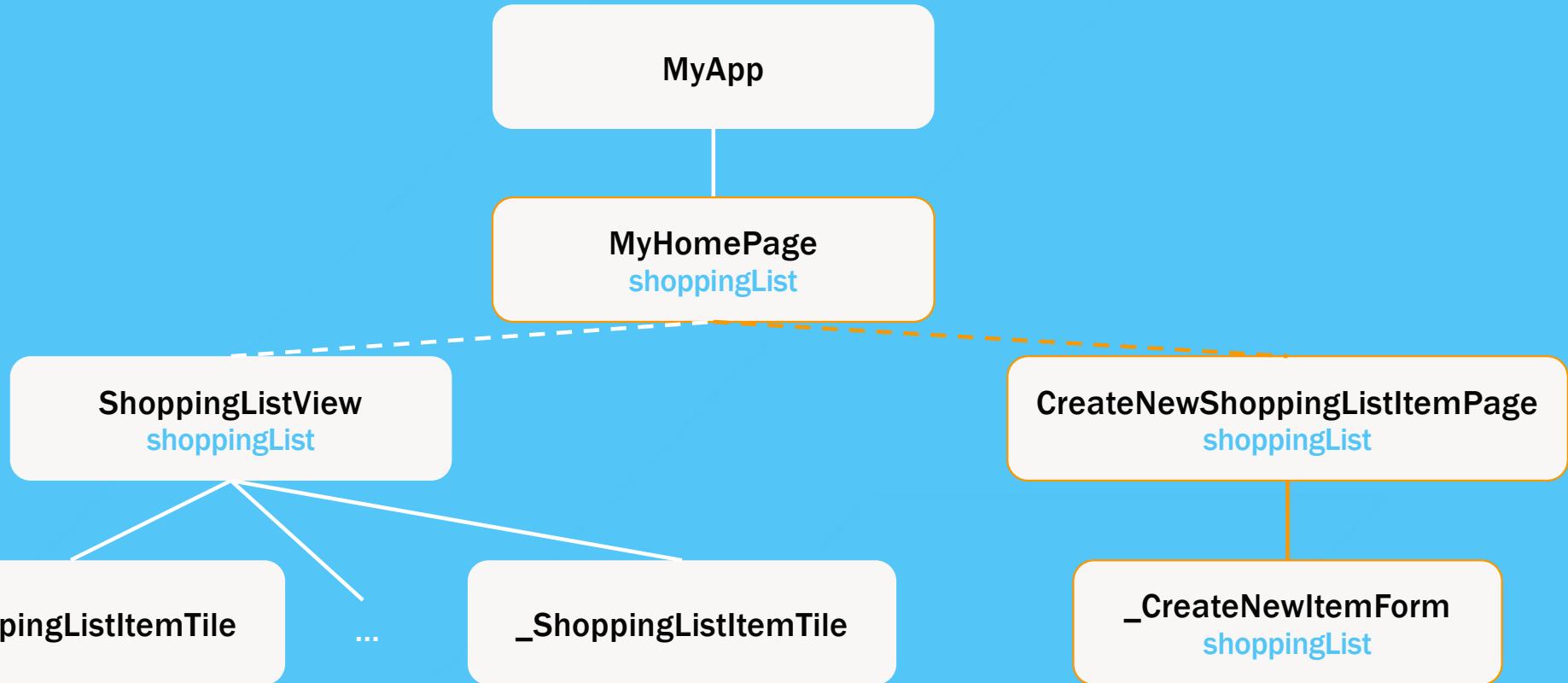


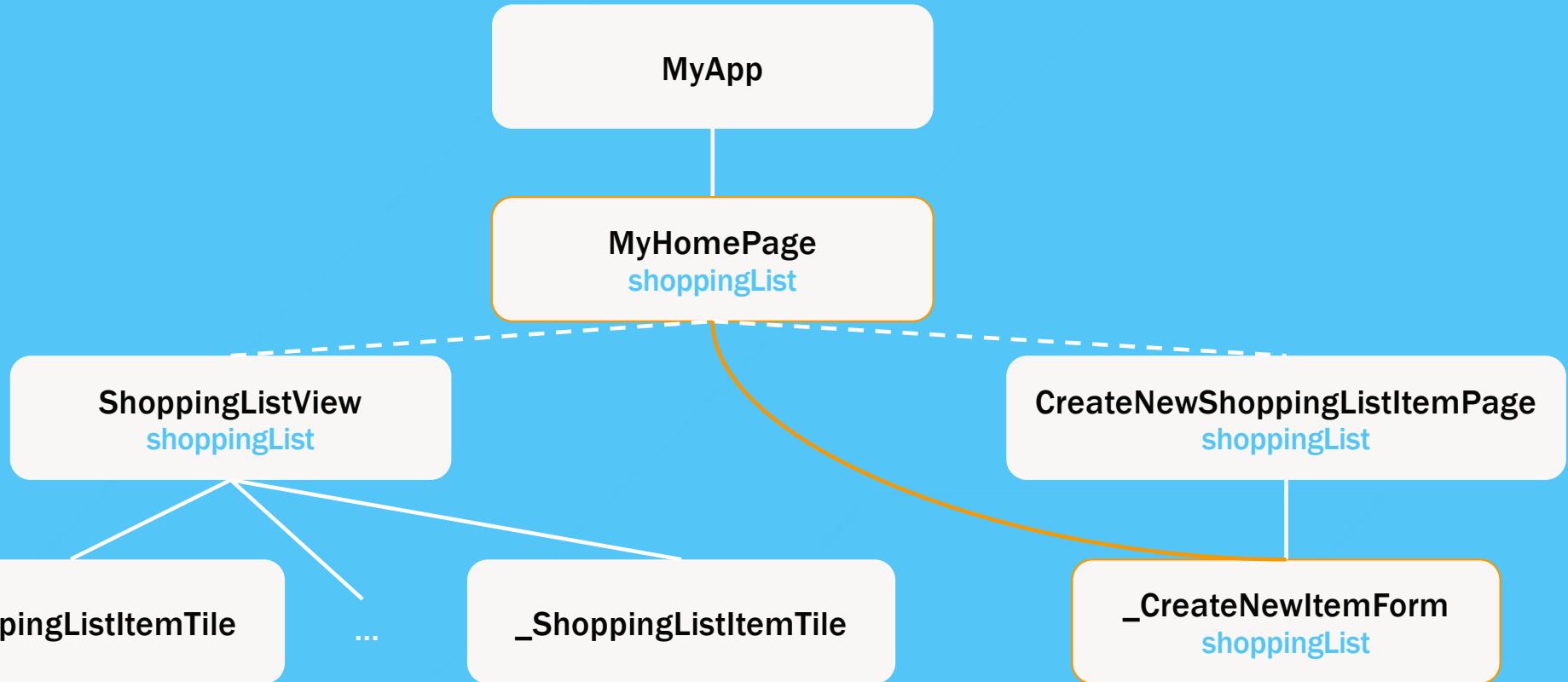






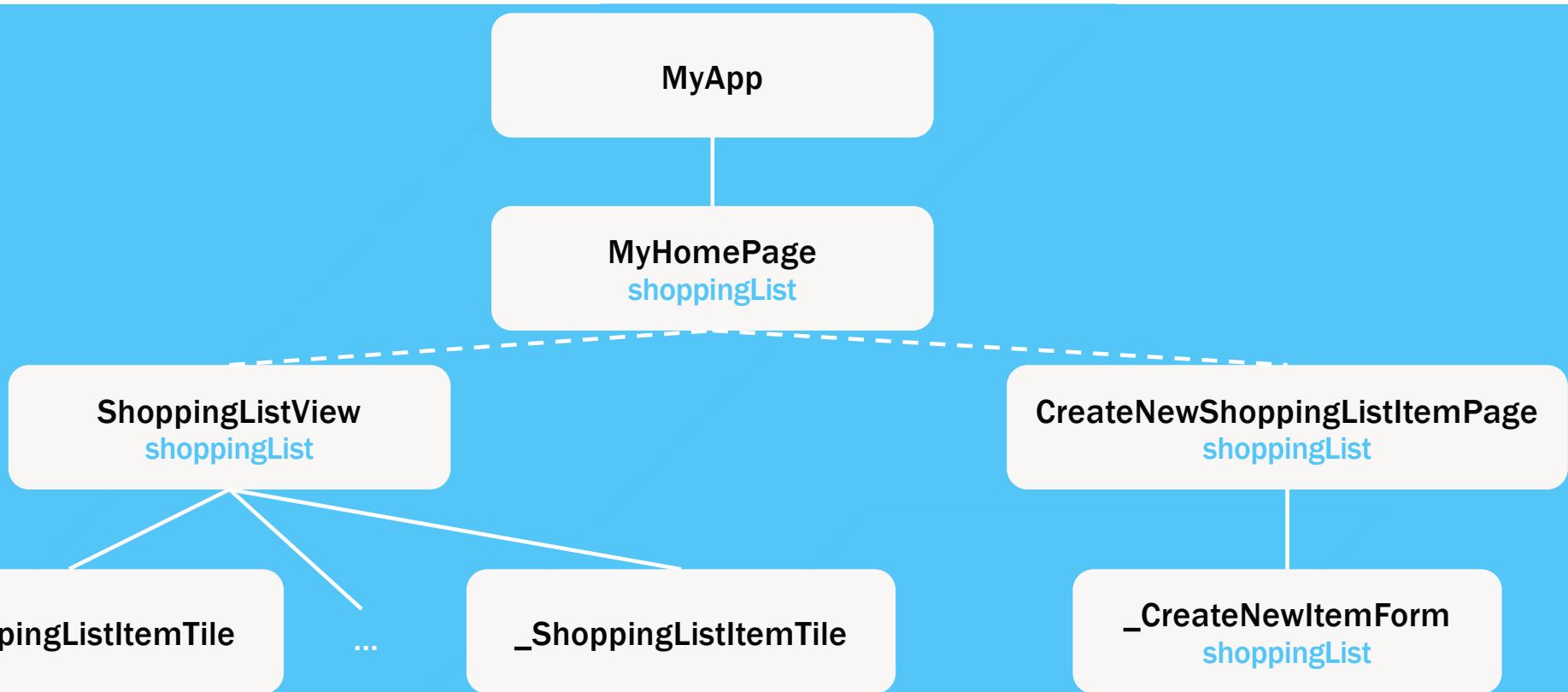


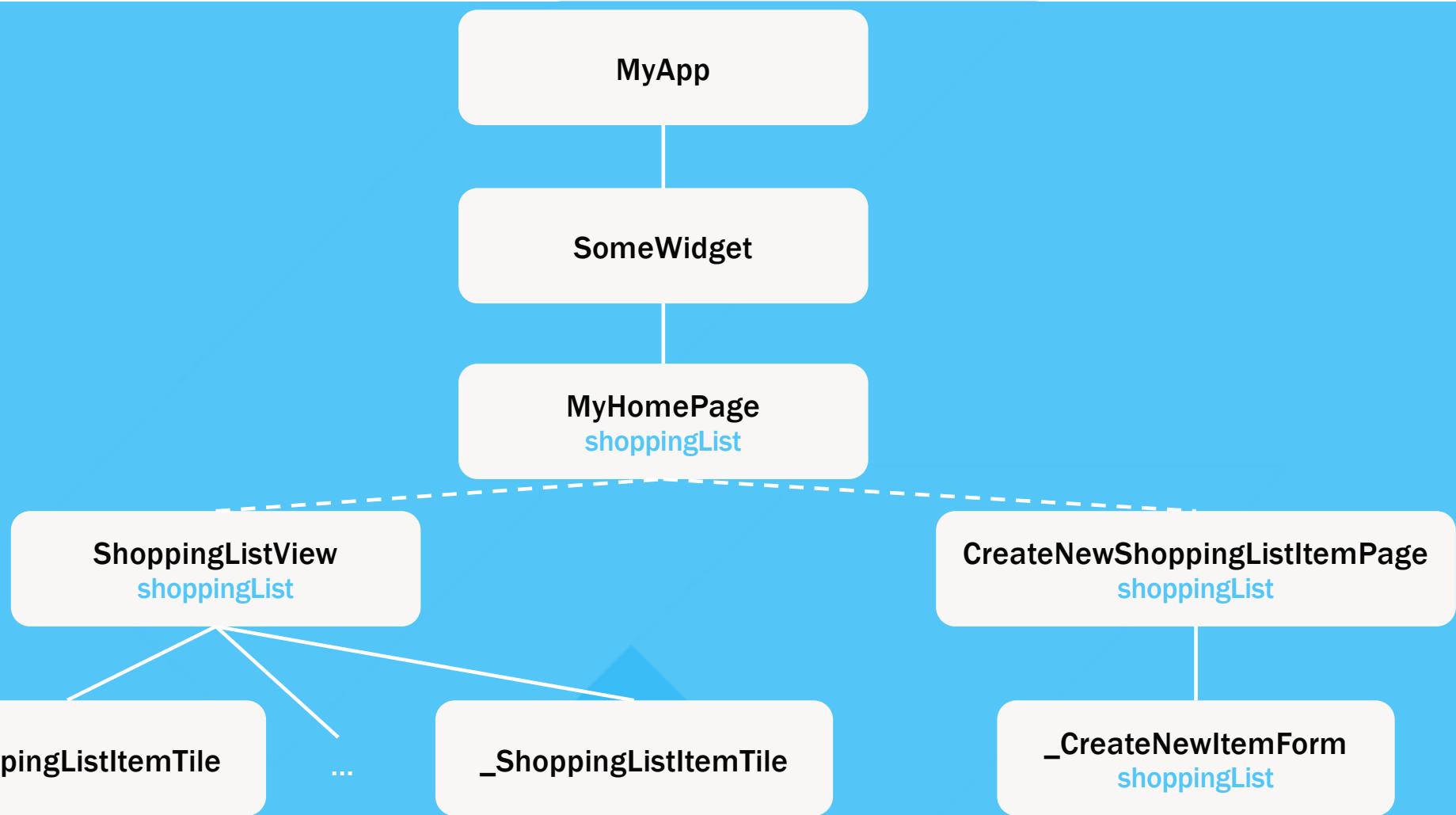




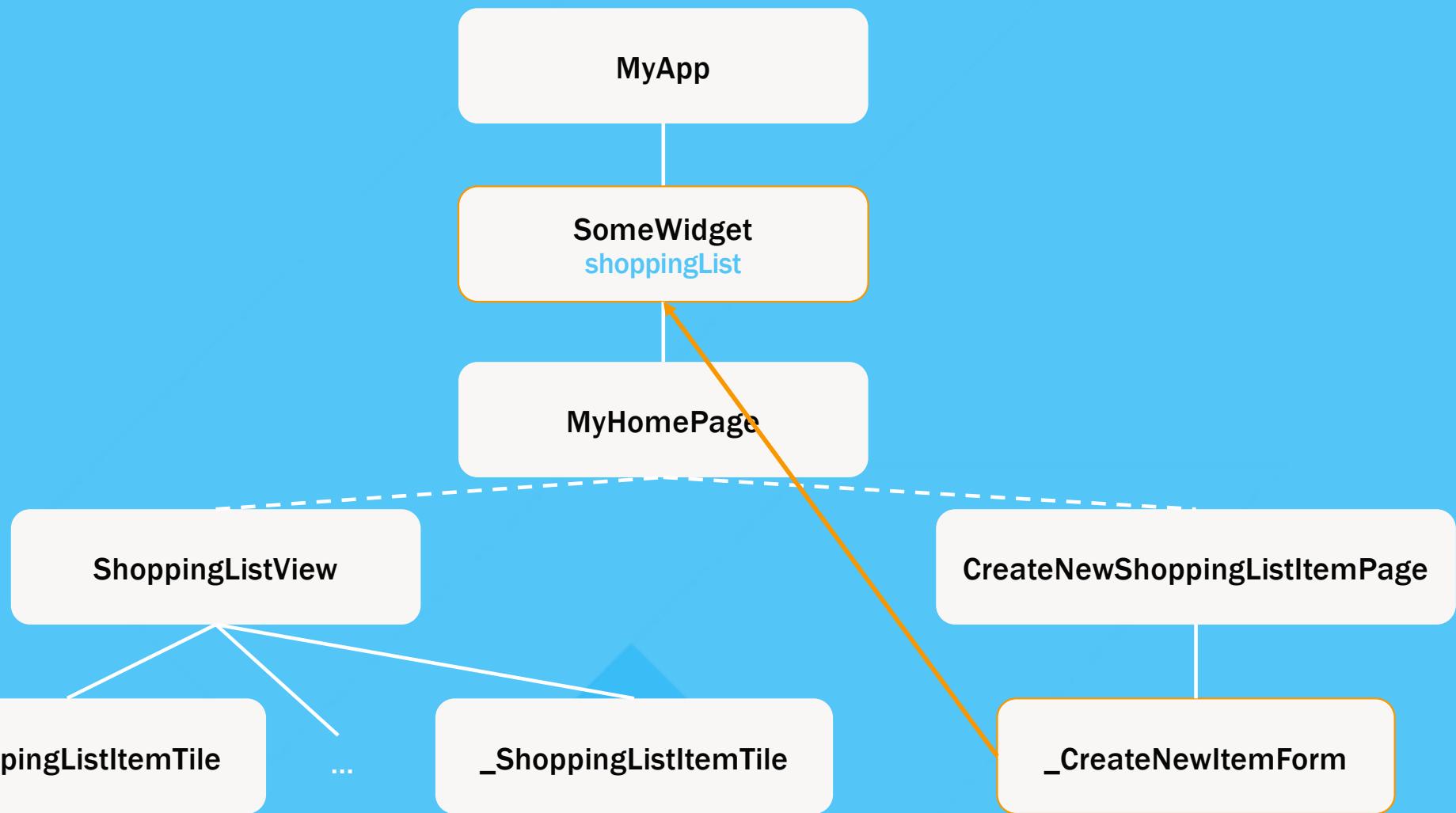
Advanced state management

The idea









Advanced state management

Options



Provider



Provider



bloc



Provider



riverpod



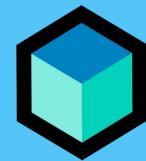
bloc



Provider



riverpod



bloc



redux



Provider



riverpod



GetX



bloc



redux



Provide



riverpod



GetX



New Flutter Devs:

Can you tell what's the best state management option for flutter ?

Flutter Team :



No, I don't think i will

Too many options