Getting Started with Flutter

Jumail Bin Taliba
School of Computing, UTM
March 2020

Outline

- Creating Flutter Projects
- Flutter Project Structure
- Building UI by example

Several ways of creating Flutter projects

- From VS Code
- Using Flutter tool from Command Line (Git Bash, Cmd or Powershell)
- Clone from an existing project

Creating Flutter Project from VS Code

1. In VS Code, open Command Palette, `Ctrl Shift P`
2. Choose from the menu, `Flutter New Project`

Creating Flutter Project with CLI

1. Open `Git Bash`
2. Move to the directory where you save all your flutter projects.
   ```
   cd d:/code/flutter
   ```
3. Create a new directory in it for your new project
   ```
   mkdir flutter_counter
   ```
4. Move to the new project directory
   ```
   cd flutter_counter
   ```
5. Run flutter tool to start creating the project *(don't forget the dot)*
   ```
   flutter create .
   ```
6. Open the source code into VS Code (*don't forget the dot*)
   ```
   code .
   ```

Creating Flutter Project by cloning

1. Copy the folder of any of your existing project.
   - You can clone from git file (created with git bundle), or from online repo (e.g. github)
   - You can copy from a zip file, or a folder

   In the following example, we are going to copy / clone from my project template on github.

2. Clone my github repo *(command below should be in one-line)*

   ```
   git clone https://github.com/jumail-utm/flutter_template.git my_project
   ```

   You can use any project name. Here, we use `my_project`

3. Move to the project directory

   ```
   cd my_project
   ```

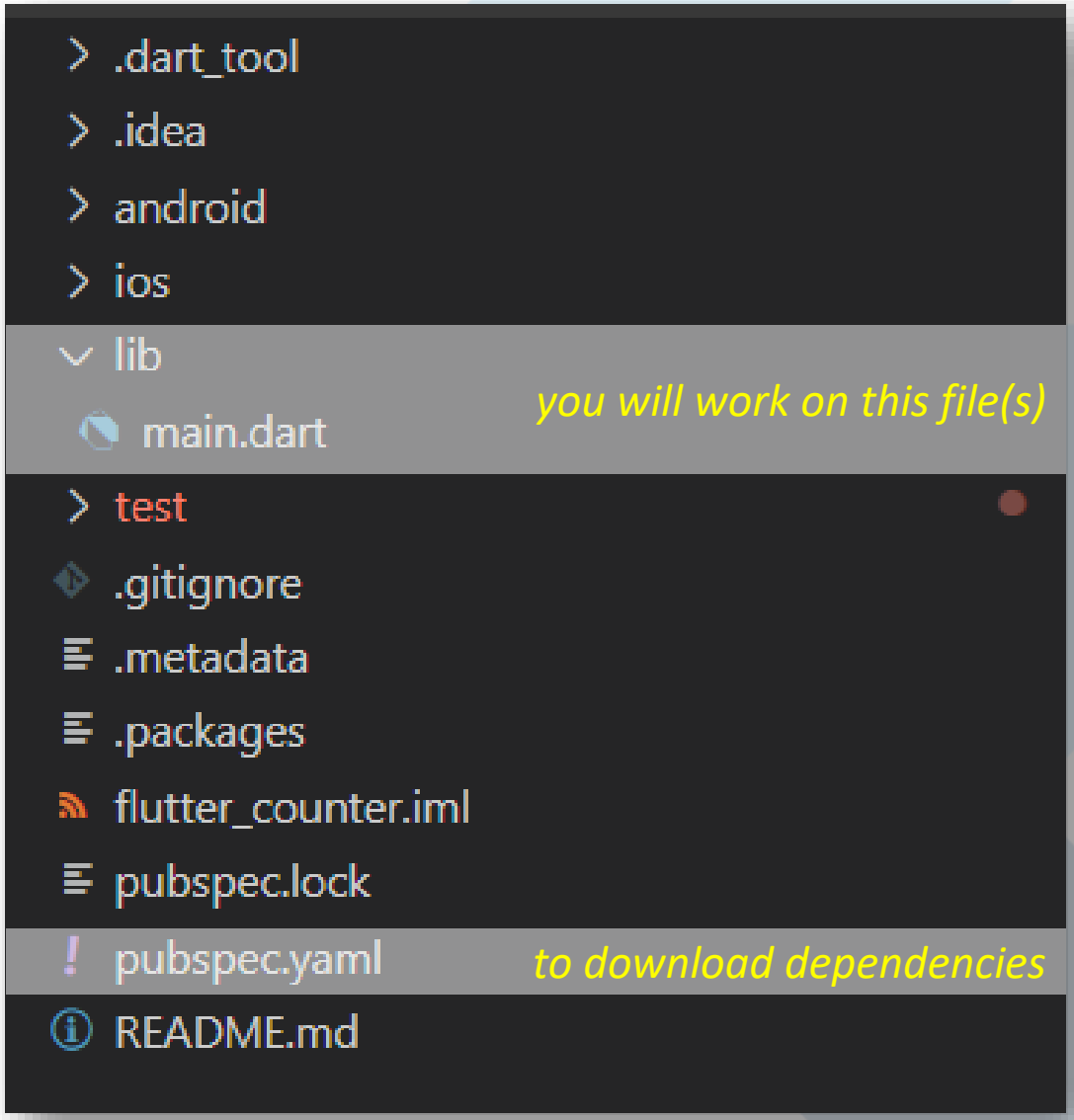
4. Run flutter tool to **properly** rename the project *(don't forget the dot)*

   ```
   flutter create -project-name .
   ```

5. Open the source code into VS Code (*don't forget the dot*)

   ```
   code .
   ```

Flutter Project Structure



Basic Flutter Program

lib > main.dart > ...

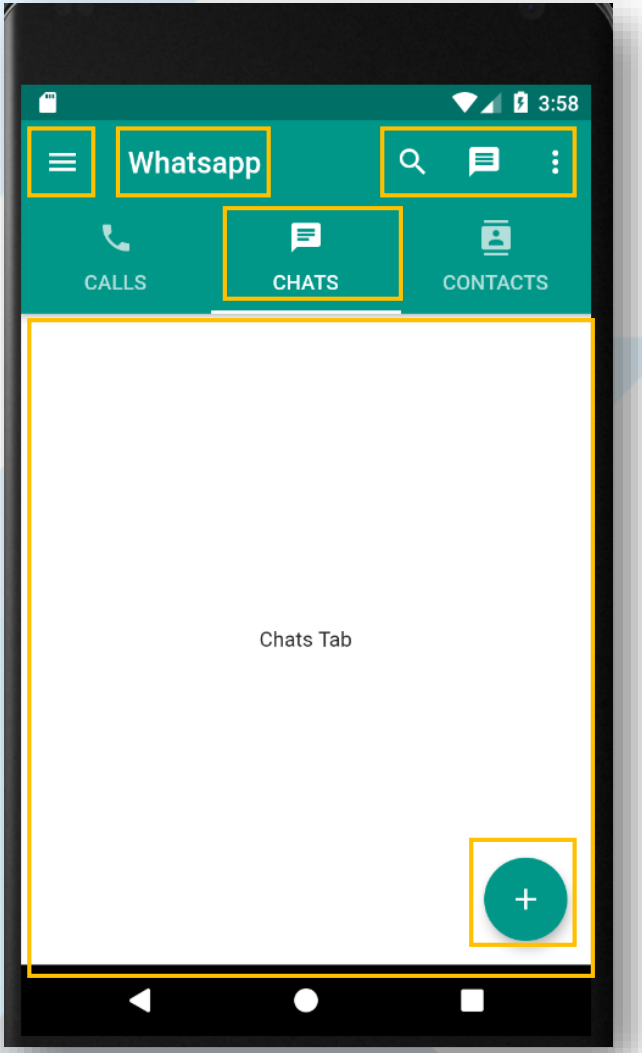
```dart
import 'package:flutter/material.dart';

void main() {
  runApp(
    MaterialApp(
      title: 'Flutter Project',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Flutter'),
        ), // AppBar
      ), // Scaffold
    ), // MaterialApp
  ); // runApp
}
```

For Convenience

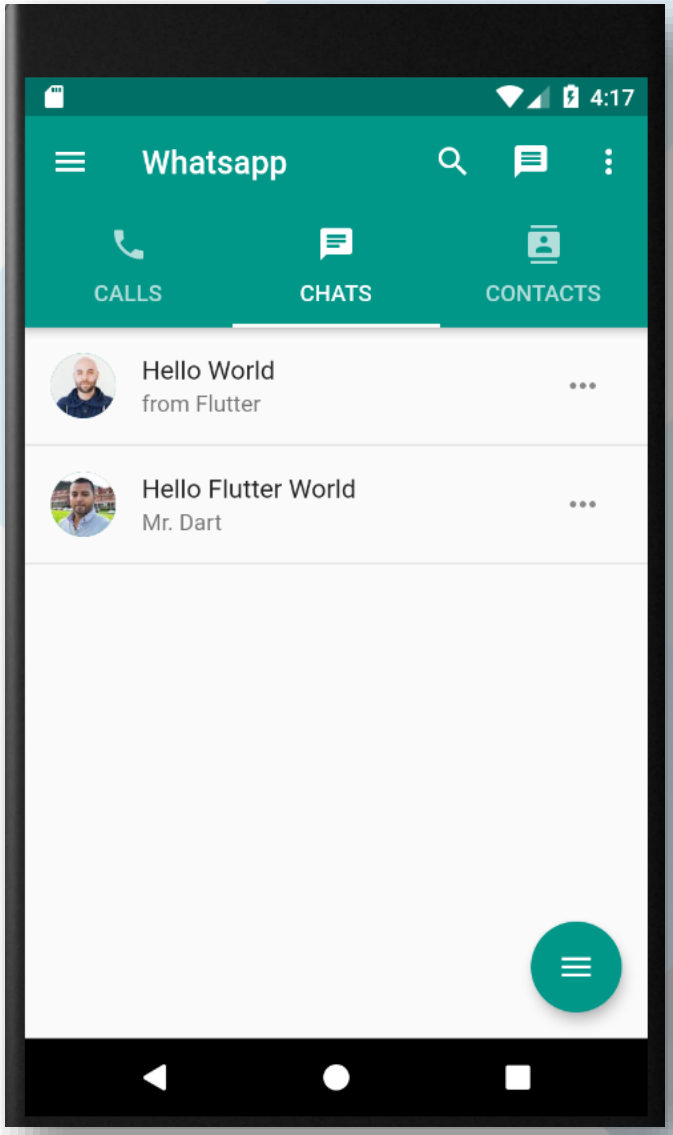
- Install VS Code extensions
  - Flutter
  - Enable: UI guides
  - Bracket Pair Colorizer
  - Pub spec Assist
- Use USB debugging (i.e. running directly on real phone)
  - Enable USB debugging and developer mode in your phone
- Use Hot Reload or Hot Restart when debugging your code
  - Hot Reload : `Ctrl F5`
  - Hot Restart : `Ctrl Shift F5`

Introduction to Widgets

- What is a widget?
  - The building blocks that construct the application
- Widgets in Flutter are immutable
- Type of widgets:
  - Stateless – does not change at all
  - Stateful – can change to reflect the state
- Common widgets:
  - App widgets (e.g. MaterialApp)
  - Convenience widgets: Scaffold, NavigationBar, AppBar
  - Layout widgets: Column, Row, ListView
  - Text widgets: Text, RichText
  - Button widgets: IconButton, RaisedButton, FloatingActionButton
  - Graphics widgets: Image, Icon



Building UI, example: Whatsapp Clone



Preparing Starter Project (1)

1. Open `Git Bash`

2. Clone my github repo *(command below should be in one-line)*

```
git clone https://github.com/jumail-utm/flutter_whatsapp_ui
```

3. Move to the project directory

```
cd flutter_whatsapp_ui
```

4. Go to the starting point of the project and create a new branch to start editing

```
git checkout b35b0b8 -b playground
```

5. Open the source code into VS Code (*don't forget the dot*)

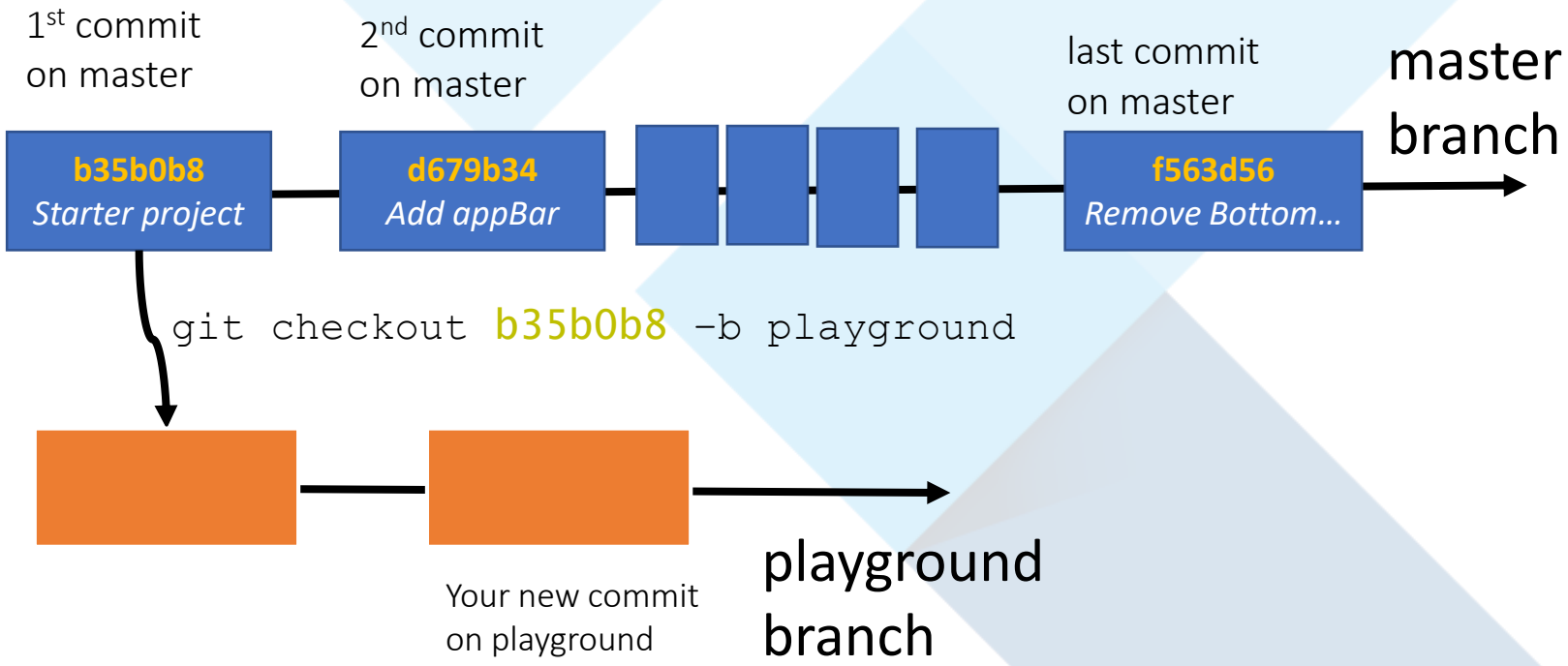
```
code .
```

6. Try running the program by pressing F5 (*you want to choose an emulator first*)

Git Basics

```
git log --oneline
```

```
f563d56 (HEAD -> master) remove BottomNavigationBar and do basic refactoring
37ac9ca add BottomNavigationBar
58ae5e6 add FloatingActionButton
362b386 add TabView
09bb5da add TabBar
d679b34 add appBar
b35b0b8 starter project
```

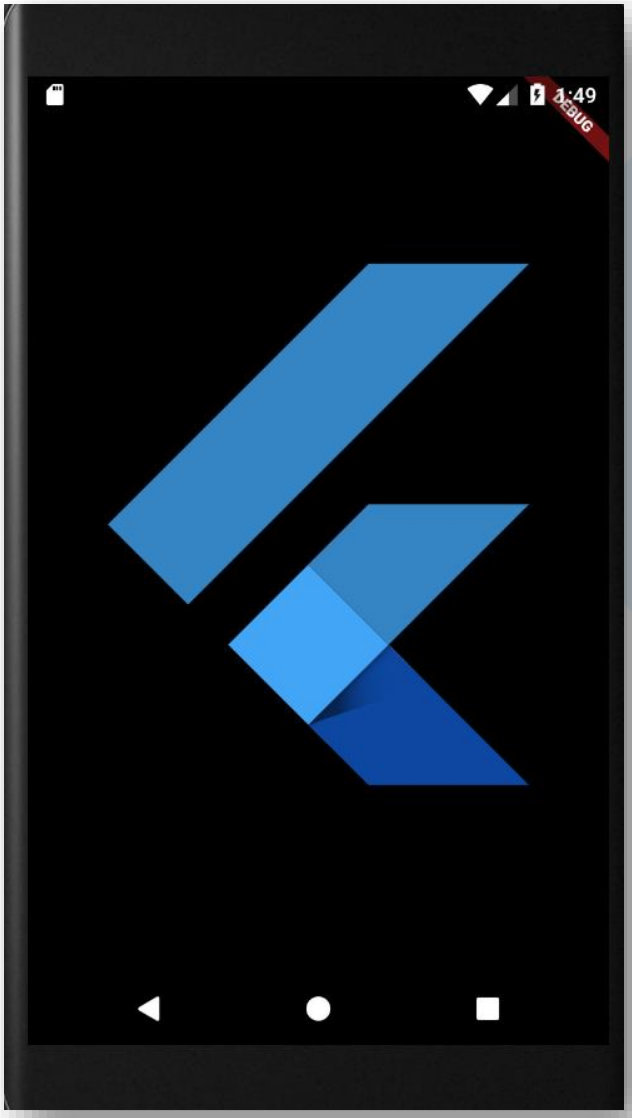


Preparing Starter Project (2)

lib > main.dart > ...

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

void main() => runApp(
      FlutterLogo(
        size: double.infinity,
      ),
    );
```

Creating a Material App



lib > main.dart > ...

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

void main() => runApp(
      MaterialApp(
        title: 'Whatsapp Clone',
        home: FlutterLogo(),
      ), // MaterialApp
    );

```

Creating a Home Screen (1)

lib > main.dart > ...

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

void main() => runApp(
        MaterialApp(
          title: 'Whatsapp Clone',
          home: Scaffold(
            body: FlutterLogo(
              size: double.infinity,
            )), // FlutterLogo // Scaffold
        ), // MaterialApp
      );
```

Creating a Home Screen (2)


```dart
void main() => runApp(
      MaterialApp(
        title: 'Whatsapp Clone',
        home: Home(),
      ), // MaterialApp
    );

class Home extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Whatsapp'),
      ), // AppBar
      body: FlutterLogo(
        size: double.infinity,
      ), // FlutterLogo
    ); // Scaffold
  }
}
```

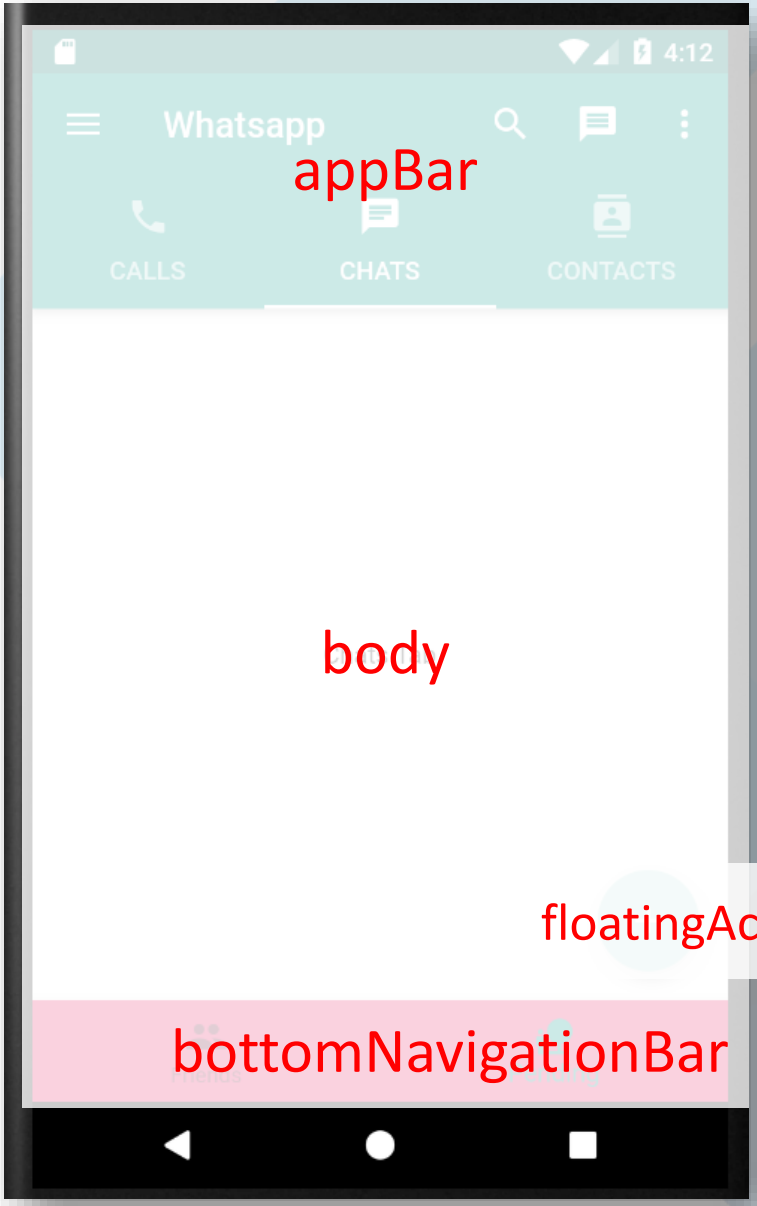
What is MaterialApp?

- A convenience widget that wraps a number of widgets commonly required for applications that embraced material design.
- It provides a bunch of things including:
  - Theming
  - Routing
  - Main route (home)
  - Localization
- MaterialApp is used to organize multiple screens for the app

Scaffold

- Provides the basic visual layout structure based on material design.
- It provides a bunch of things including:
  - appBar
  - drawer and endDrawer
  - bottomNavigationBar
  - Body
  - floatingActionButton
  - SnackBar
- Scaffold is commonly used to build UI for a screen

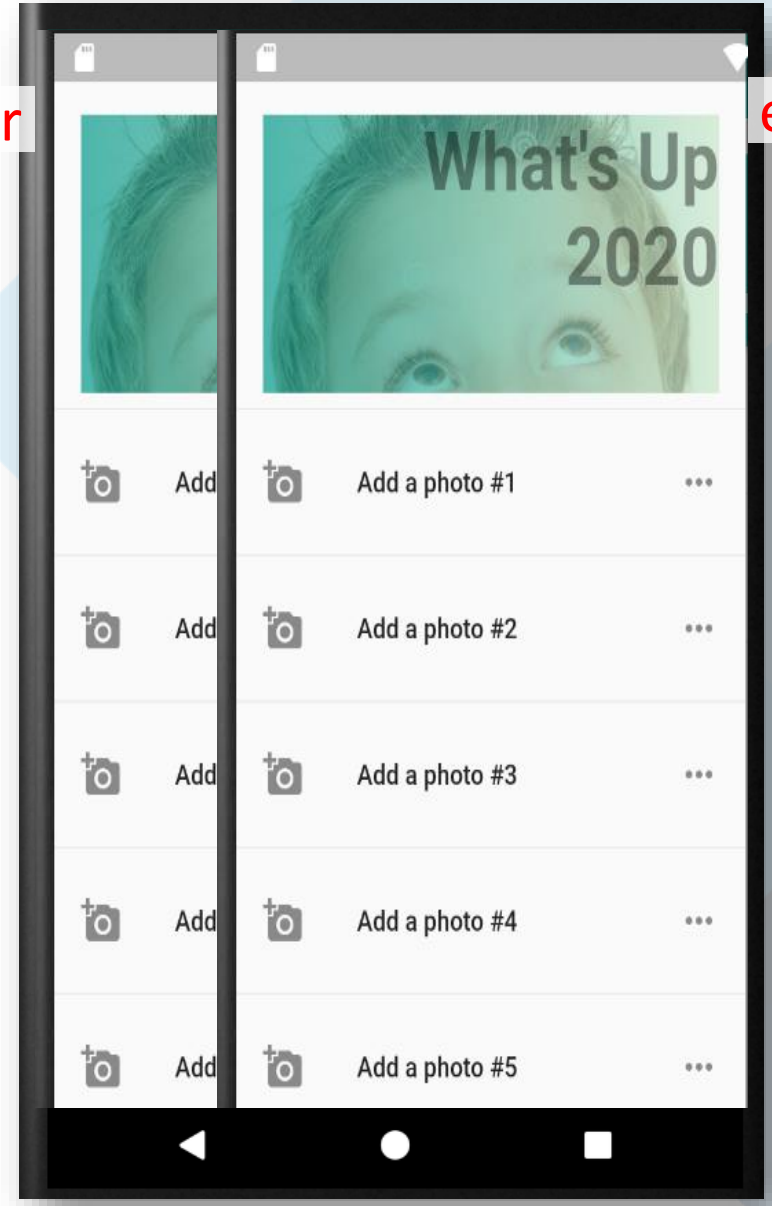
Scaffold Components (1)



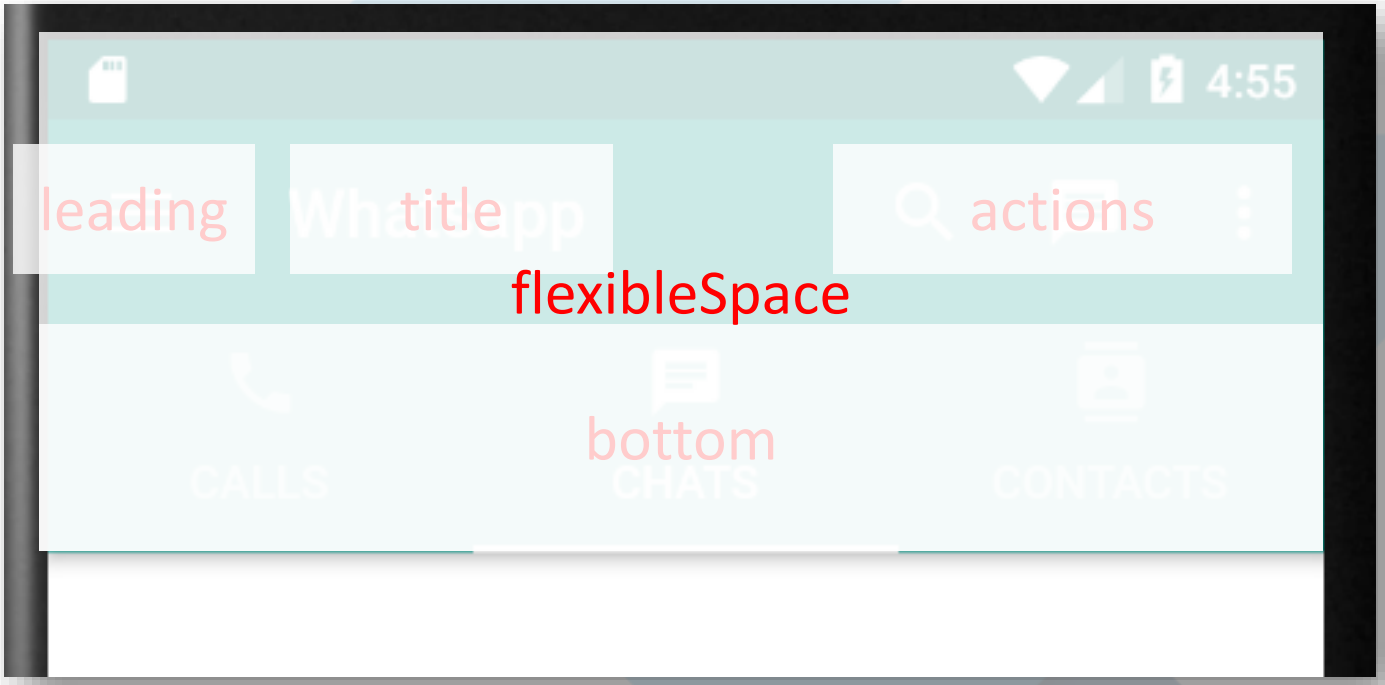
Scaffold Components (1)

drawer

endDrawer



AppBar Components

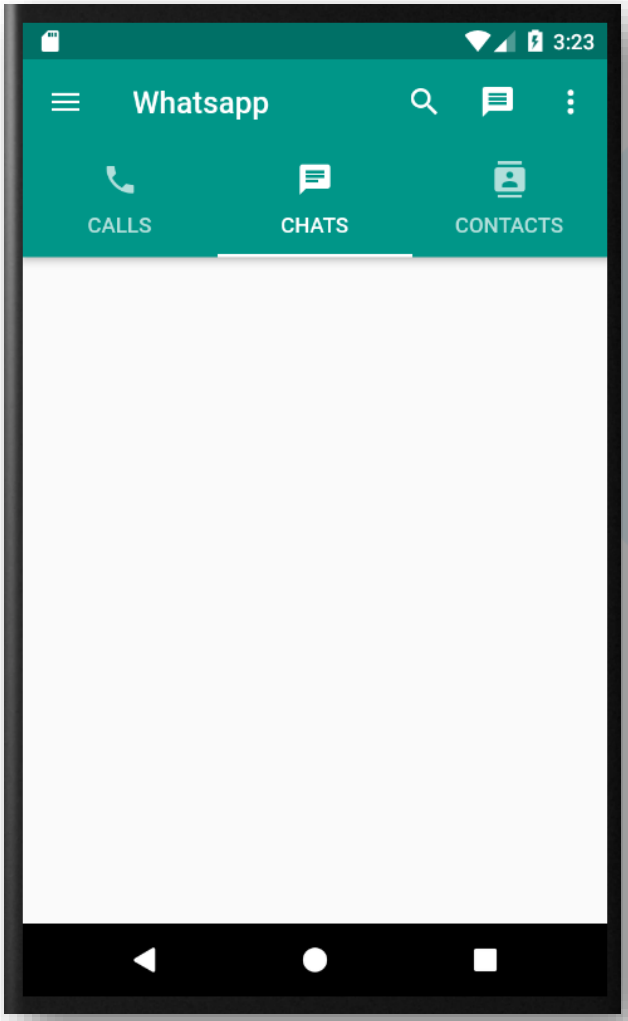


Adding AppBar


```dart
void main() => runApp(
  MaterialApp(
    debugShowCheckedModeBanner: false,
    title: 'Whatsapp Clone',
    theme: ThemeData(primarySwatch: Colors.teal),
    home: Home(),
  ), // MaterialApp
);
```

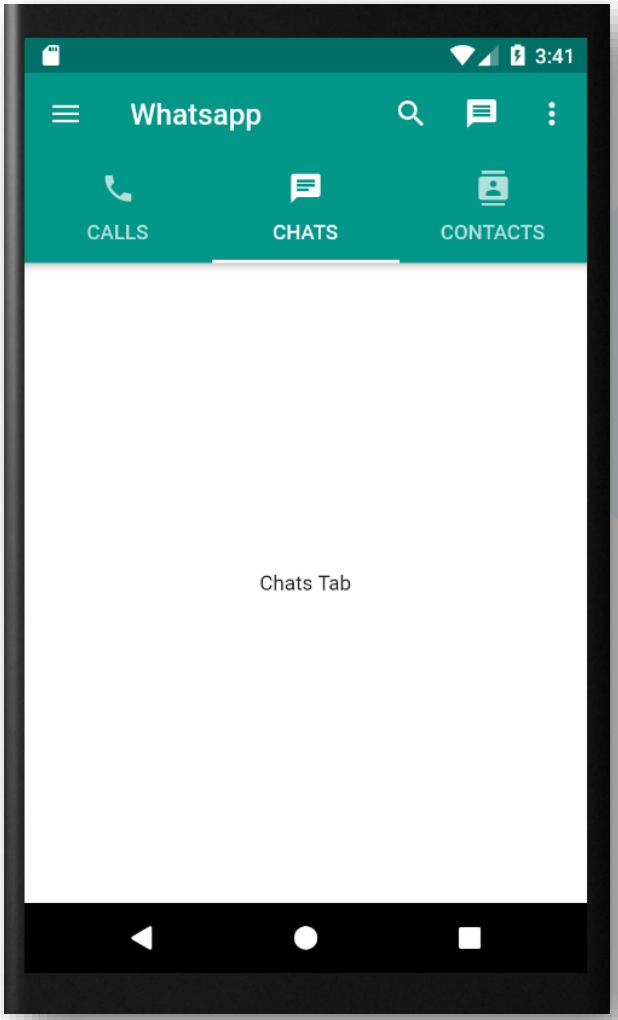
```dart
class Home extends StatelessWidget {
  void _doNothing() {}

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        leading: IconButton(
          icon: Icon(Icons.menu),
          onPressed: _doNothing,
        ), // IconButton
        title: Text('Whatsapp'),
        actions: <Widget>[
          IconButton(
            icon: Icon(Icons.search),
            onPressed: _doNothing,
          ), // IconButton
          IconButton(
            icon: Icon(Icons.message),
            onPressed: _doNothing,
          ), // IconButton
          IconButton(
            icon: Icon(Icons.more_vert),
            onPressed: _doNothing,
          ), // IconButton
        ], // <Widget>[]
      ), // AppBar
    ); // Scaffold
  }
}
```

Adding TabBar



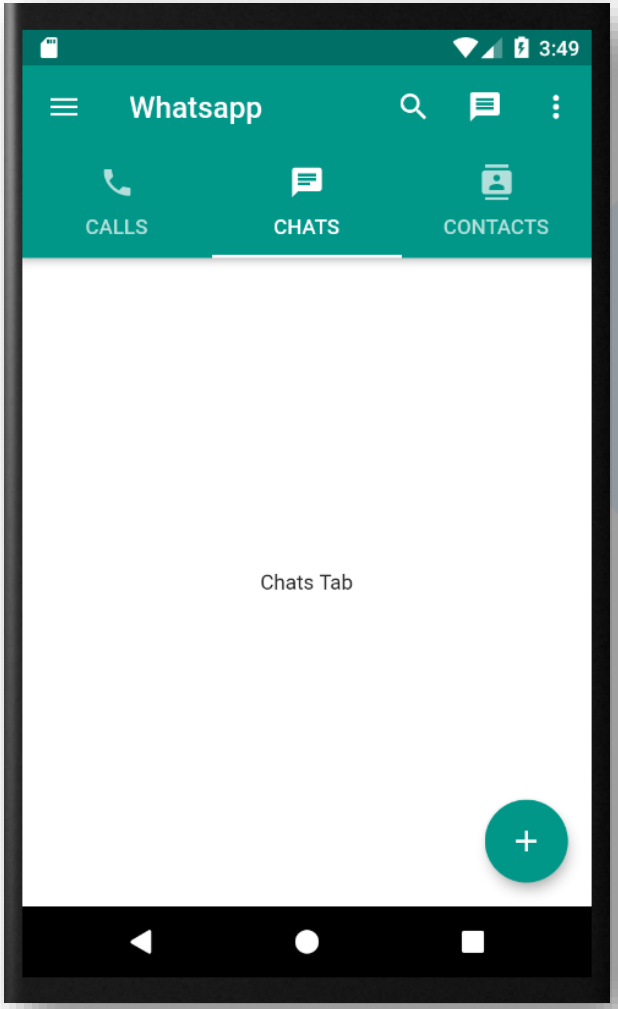

```dart
class Home extends StatelessWidget {
  void _doNothing() {}

  @override
  Widget build(BuildContext context) {
    return DefaultTabController(                    // TabBar must have TabController
      length: 3,
      initialIndex: 1,
      child: Scaffold(
        appBar: AppBar(
          leading: IconButton(
            icon: Icon(Icons.menu),
            onPressed: _doNothing,
          ), // IconButton
          title: Text('Whatsapp'),
          actions: <Widget>[
            IconButton(icon: Icon(Icons.search), onPressed: _doNothing),
            IconButton(icon: Icon(Icons.message), onPressed: _doNothing),
            IconButton(icon: Icon(Icons.more_vert), onPressed: _doNothing),
          ], // <Widget>[]
          bottom: TabBar(tabs: [
            Tab(text: 'CALLS', icon: Icon(Icons.call)),
            Tab(text: 'CHATS', icon: Icon(Icons.chat)),
            Tab(text: 'CONTACTS', icon: Icon(Icons.contacts)),
          ]), // TabBar
        ), // AppBar
      ), // Scaffold
    ); // DefaultTabController
  }
}
```

Adding TabView



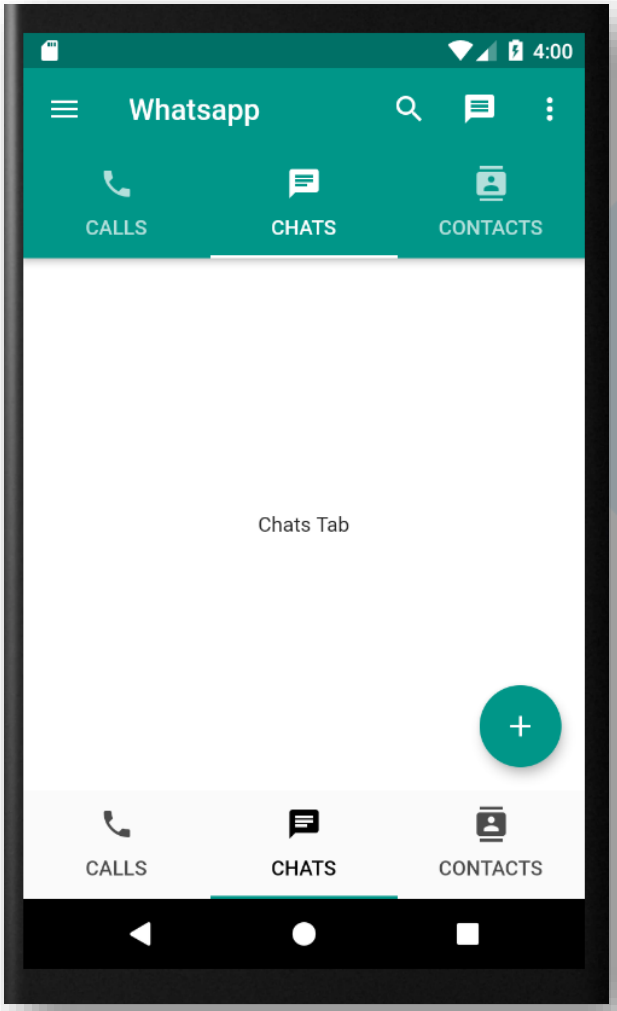

```
12  class Home extends StatelessWidget {
13    void _doNothing() {}
14
15    @override
16    Widget build(BuildContext context) {
17      return DefaultTabController(
18        length: 3,
19        initialIndex: 1,
20        child: Scaffold(
21  }       appBar: AppBar( // AppBar
38        body: TabBarView(children: [
39          Container(color: Colors.pink, child: Center(child: Text('Calls Tab'))),
40          Container(color: Colors.white, child: Center(child: Text('Chats Tab'))),
41          Container(color: Colors.amber, child: Center(child: Text('Contact Tab'))),
42        ]), // TabBarView
43      ), // Scaffold
44    ); // DefaultTabController
45    }
46  }
```

Adding FloatingActionButton



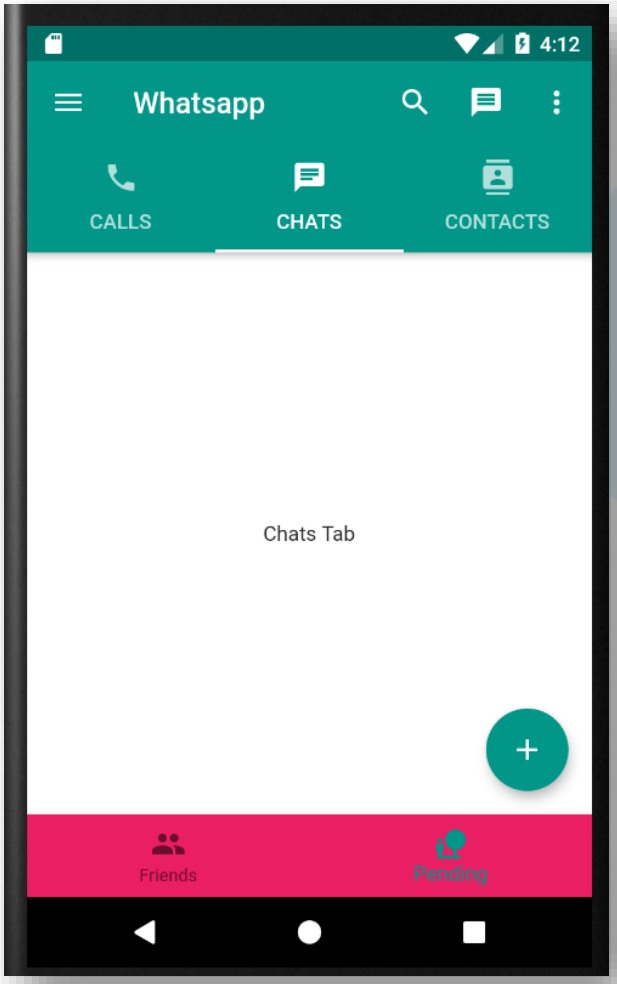

```dart
@override
Widget build(BuildContext context) {
  return DefaultTabController(
    length: 3,
    initialIndex: 1,
    child: Scaffold(
      appBar: AppBar( // AppBar
      body: TabBarView(children: [ // TabBarView
      floatingActionButton: FloatingActionButton(
        onPressed: _doNothing,
        child: Icon(Icons.add),
      ), // FloatingActionButton
    ), // Scaffold
  ); // DefaultTabController
}
}
```

bottomNavigationBar (1)



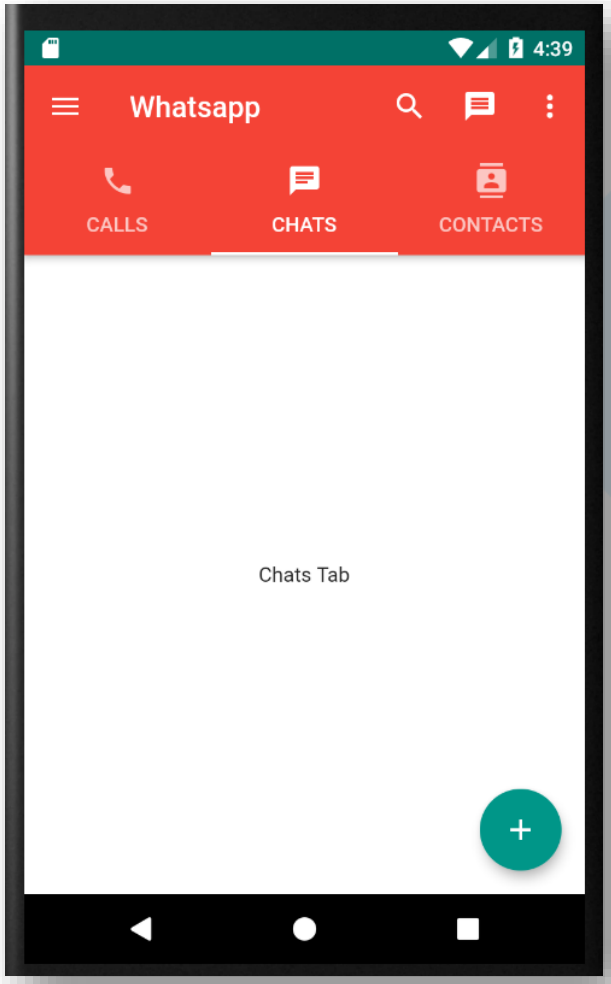

```
38  body: TabBarView(children: [ // TabBarView
46  floatingActionButton: FloatingActionButton(
47    onPressed: _doNothing,
48    child: Icon(Icons.add),
49  ), // FloatingActionButton
50  bottomNavigationBar: TabBar(tabs: [
51    Tab(text: 'CALLS', icon: Icon(Icons.call)),
52    Tab(text: 'CHATS', icon: Icon(Icons.chat)),
53    Tab(text: 'CONTACTS', icon: Icon(Icons.contacts)),
54    ], labelColor: Colors.black,), // TabBar
55
```

bottomNavigationBar (2)




```
38 ›        ├─body: TabBarView(children: [ // TabBarView
46 ›        ├─floatingActionButton: FloatingActionButton( // FloatingActionButton
50          └─bottomNavigationBar: BottomNavigationBar(
51            │ currentIndex: 1,
52            │ backgroundColor: Colors.pink,
53            │ items: [
54            │   BottomNavigationBarItem(
55            ├─────icon: Icon(Icons.people),
56            ├─────title: Text('Friends'),
57            │   ), // BottomNavigationBarItem
58            │   BottomNavigationBarItem(
59            ├─────icon: Icon(Icons.nature_people),
60            └─────title: Text('Pending'),
61              ), // BottomNavigationBarItem
62            ],
63          ), // BottomNavigationBar
64        ), // Scaffold
65      ); // DefaultTabController
66    }
67  }
```

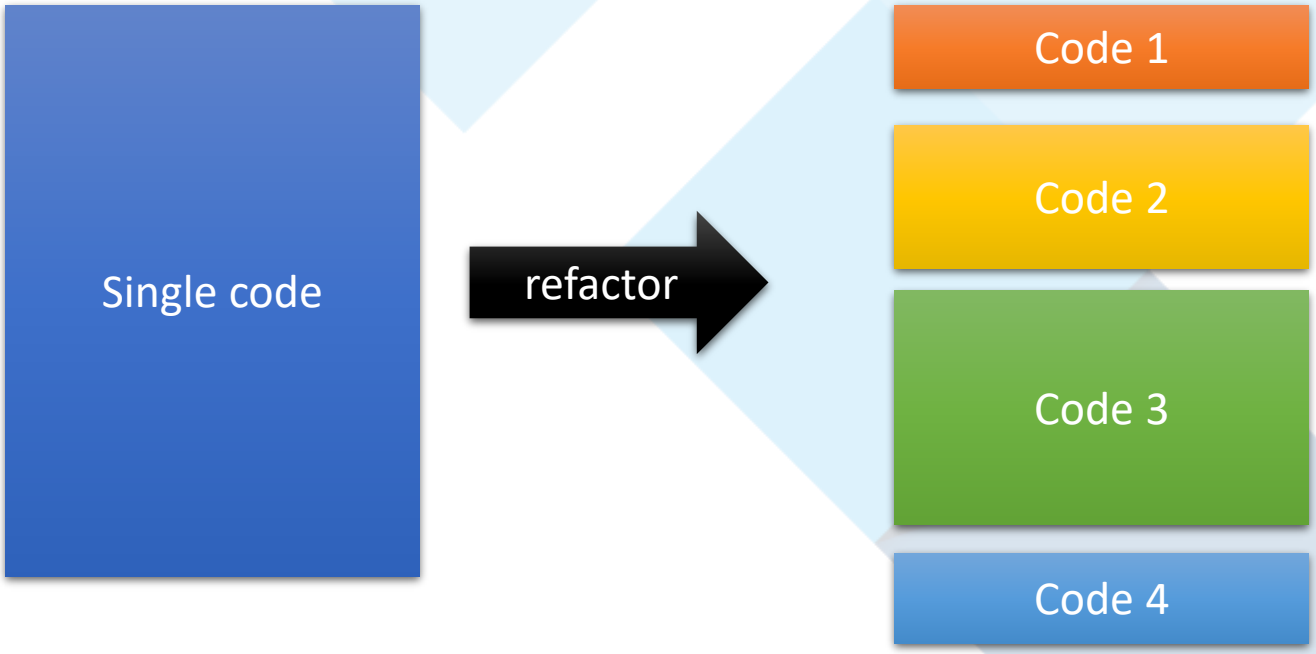
AppBar's flexibleSpace and SafeArea




```dart
Widget build(BuildContext context) {
  return DefaultTabController(
    length: 3,
    initialIndex: 1,
    child: Scaffold(
      appBar: AppBar(
        leading: IconButton(
          icon: Icon(Icons.menu),
          onPressed: _doNothing,
        ), // IconButton
        title: Text('Whatsapp'),
        actions: <Widget>[ // <Widget>[]
        bottom: TabBar(tabs: [ // TabBar
        flexibleSpace: SafeArea(
          child: Container(color: Colors.red),
        ), // SafeArea
      ), // AppBar
      body: TabBarView(children: [ // TabBarView
      floatingActionButton: FloatingActionButton(
    ), // Scaffold
  ); // DefaultTabController
}
}

```

Refactoring

- As you progress building the UI, the widget tree grows very quickly making the code less readable.
- Thus, it is important to refactor the code to make it more readable and much easier to work with.
- Basic strategy is by splitting code into several pieces of code rather than putting it in a single bulky one.



How to Refactoring

- Refactor to constants
- Refactor to methods and functions
- Refactor to classes
- Further refactoring: files and directories

How to Refactoring (1)

Refactor to constant

Delegate widget creation to variables

In VS Code:

Extract Local Variable

Before refactoring

```
20      └child: Scaffold(
21        ├appBar: AppBar(
22        │ ├leading: IconButton(
23        │ │ └icon: Icon(Icons.menu),
24        │ │   onPressed: _doNothing,
25        │ ), // IconButton
26        │ ├title: Text('Whatsapp'),
27        │ actions: <Widget>[
28        │ ├──IconButton(icon: Icon(Icons.search), onPressed: _doNothing),
29        │ ├──IconButton(icon: Icon(Icons.message), onPressed: _doNothing),
30        │ ├──IconButton(icon: Icon(Icons.more_vert), onPressed: _doNothing),
31        │ ], // <Widget>[]
32        │ └bottom: TabBar(tabs: [
33        │   ├Tab(text: 'CALLS', icon: Icon(Icons.call)),
34        │   ├Tab(text: 'CHATS', icon: Icon(Icons.chat)),
35        │   └Tab(text: 'CONTACTS', icon: Icon(Icons.contacts)),
36        │   ]), // TabBar
37        ), // AppBar
38        ├body: TabBarView(children: [
```

After refactoring

```
16    Widget build(BuildContext context) {
17      final _appBar = AppBar(
18    >   ├leading: IconButton( // IconButton
22        ├title: Text('Whatsapp'),
23    >   │ actions: <Widget>[ // <Widget>[]
28    >   └bottom: TabBar(tabs: [ // TabBar
33        ); // AppBar
34
35        return DefaultTabController(
36          length: 3,
37          initialIndex: 1,
38          └child: Scaffold(
39            ├appBar: _appBar,
40    >       ├body: TabBarView(children: [ // TabBarView
48    >       └floatingActionButton: FloatingActionButton(
```

How to Refactoring (2)

Refactor to method
Split code within the class

In VS Code:
Extract Method

After refactoring

```
15    @override
16    Widget build(BuildContext context) {
17      return DefaultTabController(
18        length: 3,
19        initialIndex: 1,
20        child: Scaffold(
21          appBar: _buildAppBar(),
22          body: TabBarView(children: [ // TabBarView
30          floatingActionButton: FloatingActionButton( /
34        ), // Scaffold
35      ); // DefaultTabController
36    }
37
38    AppBar _buildAppBar() {
39      return AppBar(
40        leading: IconButton( // IconButton
44        title: Text('Whatsapp'),
45        actions: <Widget>[ // <Widget>[]
50        bottom: TabBar(tabs: [ // TabBar
55      ); // AppBar
56    }
57  }
```

How to Refactoring (3)

Refactor to function

Split code to outside of class

In VS Code:

Extract Method

After refactoring

```
12  class Home extends StatelessWidget {
13    void _doNothing() {}
14
15    @override
16    Widget build(BuildContext context) {
17      return DefaultTabController(
18        length: 3,
19        initialIndex: 1,
20        child: Scaffold(
21          appBar: _buildAppBar(),
22          body: TabBarView(children: [ // TabBarView
30          floatingActionButton: FloatingActionButton(
34        ), // Scaffold
35      ); // DefaultTabController
36    }
37  }
38
39  void _doNothing() {}
40
41  AppBar _buildAppBar() {
42    return AppBar(
43      leading: IconButton( // IconButton
47      title: Text('Whatsapp'),
48      actions: <Widget>[ // <Widget>[]
53      bottom: TabBar(tabs: [ // TabBar
58    ); // AppBar
59  }
```

How to Refactoring (4)

Refactor to class

Split code to different classes.
It iRecommended approach

In VS Code:

Extract Widget

After refactoring

```
12  class Home extends StatelessWidget {
13    void _doNothing() {}
14    @override
15    Widget build(BuildContext context) {
16      return DefaultTabController(
17        length: 3,
18        initialIndex: 1,
19        child: Scaffold(
20          appBar: _AppBar(),
21          body: TabBarView(children: [ // TabBarView
29          floatingActionButton: FloatingActionButton(
33        ), // Scaffold
34      ); // DefaultTabController
35    }
36  }
37
38  class _AppBar extends StatelessWidget
39                implements PreferredSizeWidget {
40    void _doNothing() {}
41    const _AppBar({ Key key,}) : super(key: key);
42    @override
43    Size get preferredSize => Size.fromHeight(100);
44
45    @override
46    Widget build(BuildContext context) {
47      return AppBar(
48        leading: IconButton( // IconButton
52        title: Text('Whatsapp*'),
53        actions: <Widget>[ // <Widget>[]
58        bottom: TabBar(tabs: [ // TabBar
63      ); // AppBar
64    }
65  }
```

How to Refactoring (5)

- Further refactoring to separate files
- It is a good practice to use dedicated files and directory structure
  - Easy to maintain
  - Easy to scale
  - Better for teamwork
- Example project directory structure:

```
[project_root_dir]
        |
        +---[assets]
        |       + ---[images]
        |       |        + ---photo.jpg
        |       |        + ---background.png
        |       |
        |       + ---[fonts]
        |
        +---[backends]
        |       + ---[firebase_functions]
        |       + ---[nodejs_code]
        |       + ---[php_code]
        |
        +---[lib]
        |       + ---[models]
        |       |       + ---user.dart
        |       |       + ---chat.dart
        |       |       + ---call.dart
        |       + ---[services]
        |       + ---[widgets]
        |       + ---[screens]
        |               + ---[home]
        |               |      + ---index.dart
        |               |      + ---app_bar.dart
        |               |      + ---[tabs]
        |               |              + ---chat_tab_view.dart
        |               |              + ---call_tab_view.dart
        |               |              + ---contact_tab_view.dart
        |               + ---[login]
        |               + ---[splash]
```

Refactoring Our Code (1)

Here, we are using Refactoring to functions

After refactoring

```
12  class Home extends StatelessWidget {
13    @override
14    Widget build(BuildContext context) {
15      return DefaultTabController(
16        length: 3,
17        initialIndex: 1,
18        child: Scaffold(
19          appBar: _buildAppBar(),
20          body: _buildTabBarView(),
21          floatingActionButton: _buildFloatingActionButton(),
22        ), // Scaffold
23      ); // DefaultTabController
24    }
25  } // class Home
26
```

Refactoring Our Code (2)

Refactoring AppBar

```
27  void _doNothing() {}
28
29  AppBar _buildAppBar() {
30    return AppBar(
31      leading: IconButton(
32        icon: Icon(Icons.menu),
33        onPressed: _doNothing,
34      ), // IconButton
35      title: Text('Whatsapp'),
36      actions: <Widget>[
37        IconButton(icon: Icon(Icons.search), onPressed: _doNothing),
38        IconButton(icon: Icon(Icons.message), onPressed: _doNothing),
39        IconButton(icon: Icon(Icons.more_vert), onPressed: _doNothing),
40      ], // <Widget>[]
41      bottom: TabBar(tabs: [
42        Tab(text: 'CALLS', icon: Icon(Icons.call)),
43        Tab(text: 'CHATS', icon: Icon(Icons.chat)),
44        Tab(text: 'CONTACTS', icon: Icon(Icons.contacts)),
45      ]), // TabBar
46    ); // AppBar
47  }
48
```

Refactoring Our Code (3)

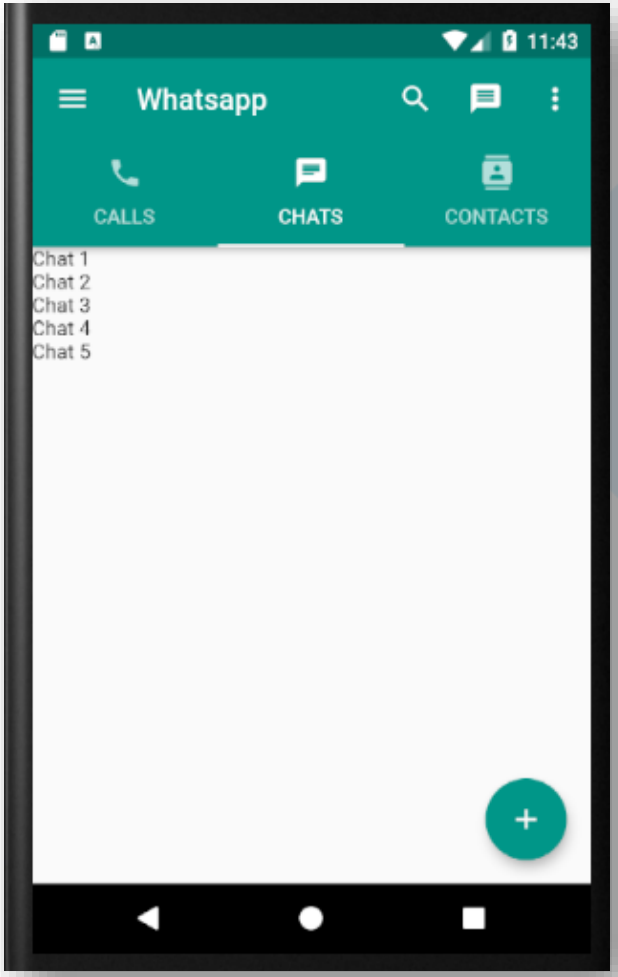
Refactoring TabBarView and FloatingActionButton

```
49  TabBarView _buildTabBarView() {
50    return TabBarView(children: [
51    ├─Container(color: Colors.pink, child: Center(child: Text('Calls Tab'))),
52    ├─Container(color: Colors.white, child: Center(child: Text('Chats Tab'))),
53    └─Container(color: Colors.amber, child: Center(child: Text('Contact Tab'))),
54    ]); // TabBarView
55  }
56
57  FloatingActionButton _buildFloatingActionButton() {
58    return FloatingActionButton(
59    │ onPressed: _doNothing,
60    └─child: Icon(Icons.add),
61    ); // FloatingActionButton
62  }
```

Several ways to add ListView

- ListView (….)
- ListView with collection-for
- ListView.builder (….)

Adding ListView – Approach 1, ListView()

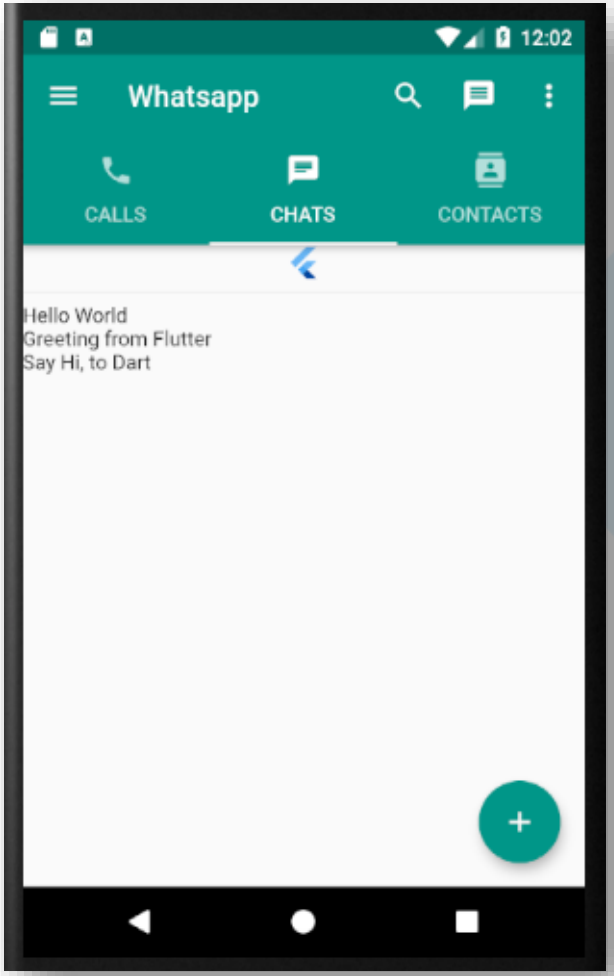


Add to CHATS Tab, in function `_buildTabBarView()`

```
49  TabBarView _buildTabBarView() {
50    return TabBarView(children: [
51    ├─Container(color: Colors.pink, child: Center(child: Text('Calls Tab'))),
52    ├─_ChatsTabView(),
53    └─Container(color: Colors.amber, child: Center(child: Text('Contact Tab'))),
54    ]);
55  }
56
57  class _ChatsTabView extends StatelessWidget {
58    @override
59    Widget build(BuildContext context) {
60      return ListView(
61        children: <Widget>[
62        ├──Text('Chat 1'),
63        ├──Text('Chat 2'),
64        ├──Text('Chat 3'),
65        ├──Text('Chat 4'),
66        └──Text('Chat 5'),
67        ],
68      );
69    }
70  }
71
```

This approach is suitable for list with known size

Adding ListView – Approach 2, with collection-for

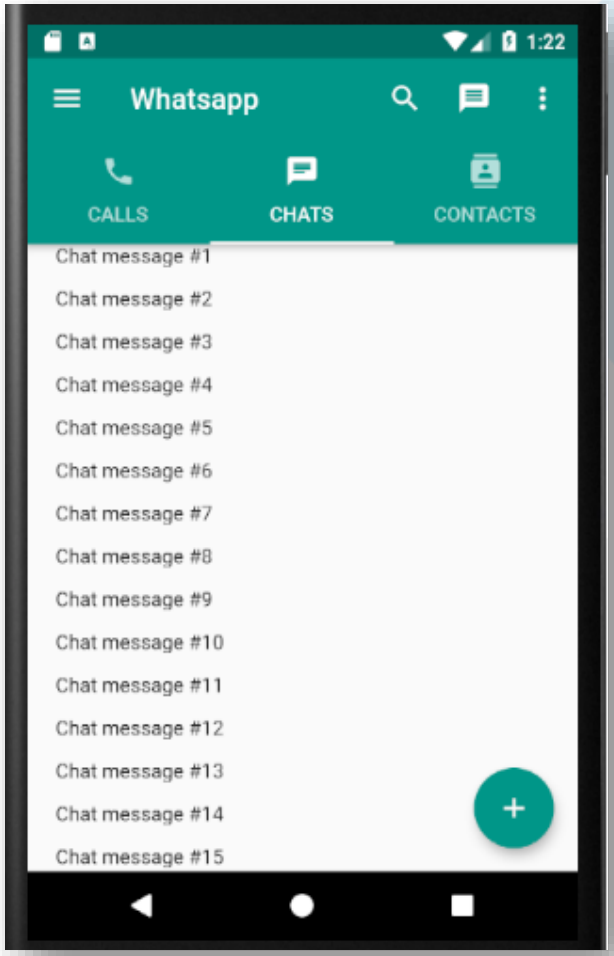


This approach can be used for dynamic list

```
49  TabBarView _buildTabBarView() {
50    return TabBarView(children: [
51    ├─Container(color: Colors.pink, child: Center(child: Text('Calls Tab'))),
52    ├─_ChatsTabView(
53    │   ['Hello World', 'Greeting from Flutter', 'Say Hi, to Dart'],
54    ),
55    └─Container(color: Colors.amber, child: Center(child: Text('Contact Tab'))),
56    ]);
57  }
58
59  class _ChatsTabView extends StatelessWidget {
60    final List items;
61
62    _ChatsTabView(this.items);
63
64    @override
65    Widget build(BuildContext context) {
66      return ListView(
67        children: <Widget>[
68        ├──FlutterLogo(),
69        ├──Divider(),
70        └──for (var item in items) Text(item),    The for, if, and spread elements w
71        ],
72      );
73    }
74  }
```

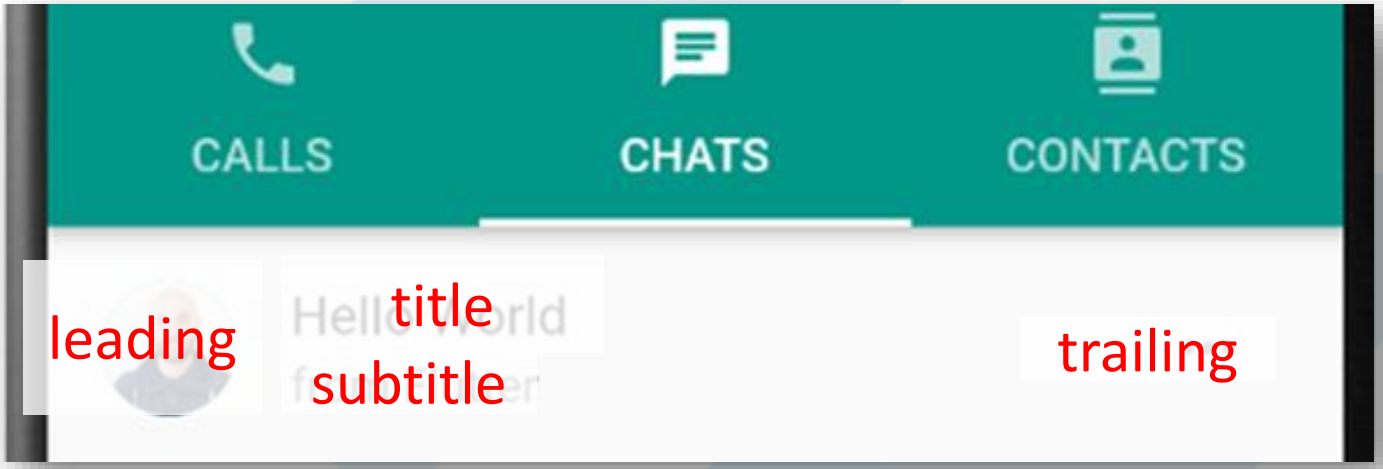
Adding ListView – Approach 3, ListView.builder()

- Another approach for dynamic list.
- It is recommended approach.
- Suitable for large list
- Removing `itemCount` will create an infinite list

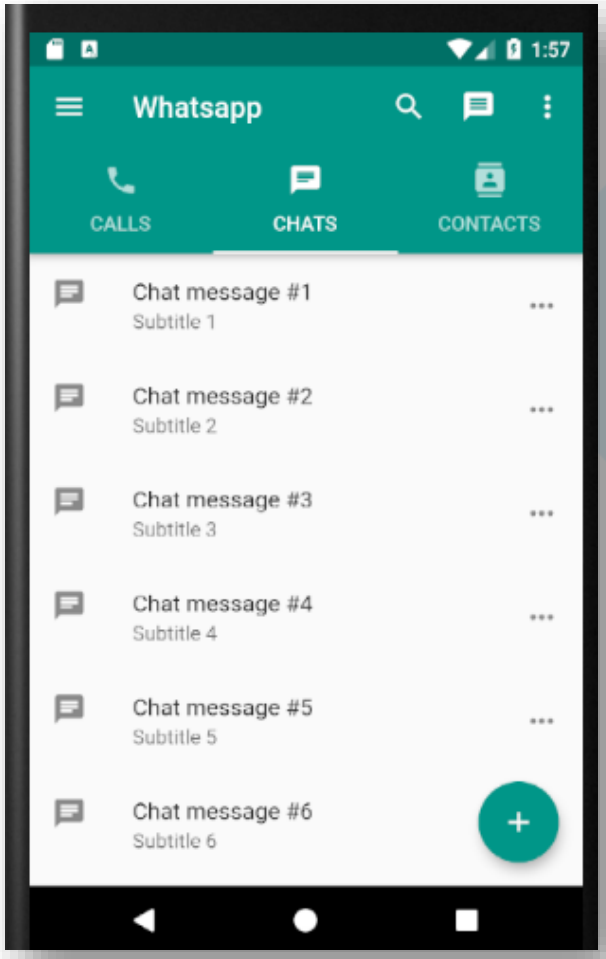



```dart
TabBarView _buildTabBarView() {
  final chats = List<String>.generate(100, (i) => 'Chat message #${i + 1}');

  return TabBarView(children: [
  ├─Container(color: Colors.pink, child: Center(child: Text('Calls Tab'))),
  ├─_ChatsTabView(chats),
  └─Container(color: Colors.amber, child: Center(child: Text('Contact Tab'))),
  ]);
}

class _ChatsTabView extends StatelessWidget {
  final List items;

  _ChatsTabView(this.items);

  @override
  Widget build(BuildContext context) {
    return ListView.builder(
    │ itemExtent: 30,
    │ padding: EdgeInsets.only(left: 20.0),
    │ itemCount: items.length,
    └─itemBuilder: (context, index) => Text(items[index]),
    );
  }
}
```

Adding ListTile, the components



Adding ListTile

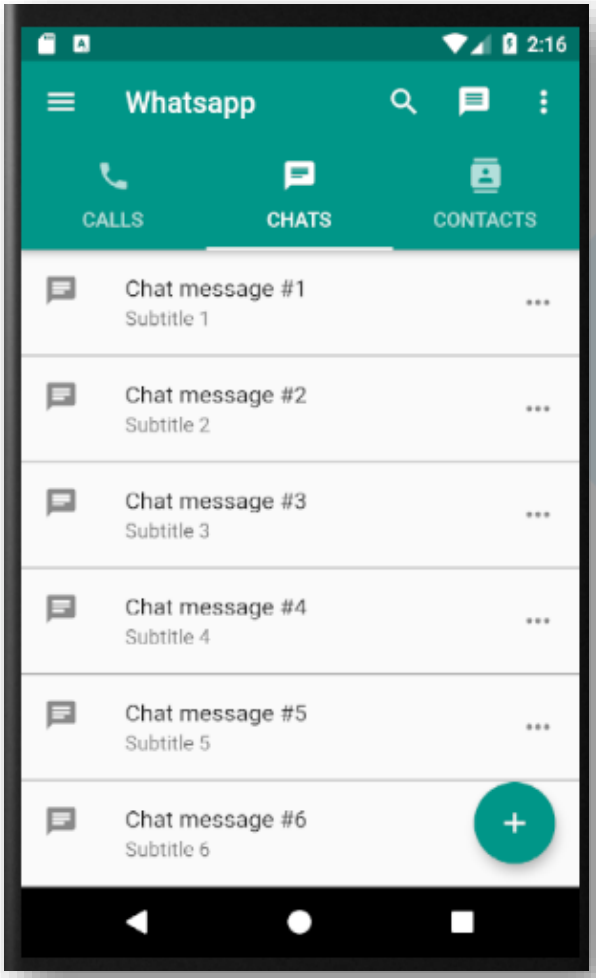



```dart
TabBarView _buildTabBarView() {
  final chats = List<String>.generate(10, (i) => 'Chat message #${i + 1}');

  return TabBarView(children: [
  ├─Container(color: Colors.pink, child: Center(child: Text('Calls Tab'))),
  ├─_ChatsTabView(chats),
  └─Container(color: Colors.amber, child: Center(child: Text('Contact Tab'))),
  ]);
}

class _ChatsTabView extends StatelessWidget {
  final List items;

  _ChatsTabView(this.items);

  @override
  Widget build(BuildContext context) {
    return ListView.builder(
      itemCount: items.length,
      itemBuilder: (context, index) => ListTile(
      ├─leading: Icon(Icons.chat),
      ├─title: Text(items[index]),
      ├─subtitle: Text('Subtitle ${index + 1}'),
      └─trailing: Icon(Icons.more_horiz),
      ),
    );
  }
}
```

Several ways to add separators

- Wrapped with Container widget and specify decoration
- Wrapped width Card widgets
- ListTile.divideTiles()
- ListView.separated (….)

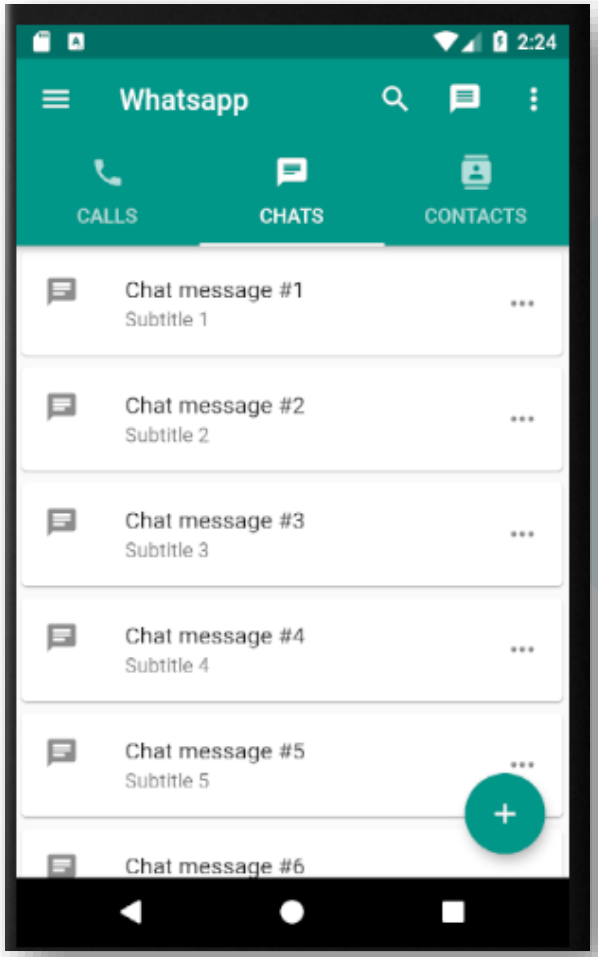
Adding Separator – Approach 1, wrapped with Container



Decorate the bottom border of the container from the attribute `decoration`

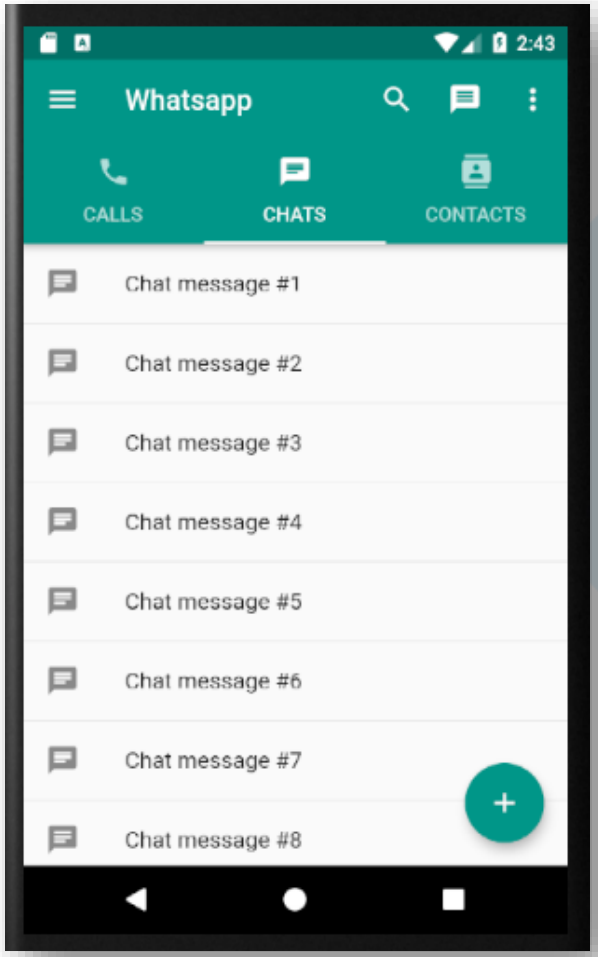
```
59  class _ChatsTabView extends StatelessWidget {
60    final List items;
61
62    _ChatsTabView(this.items);
63
64    @override
65    Widget build(BuildContext context) {
66      return ListView.builder(
67        itemCount: items.length,
68        itemBuilder: (context, index) => Container(
69          decoration: BoxDecoration(
70            border: Border(
71              bottom: BorderSide(color: Colors.grey, style: BorderStyle.solid),
72            ),
73          ),
74          child: ListTile(
75            leading: Icon(Icons.chat),
76            title: Text(items[index]),
77            subtitle: Text('Subtitle ${index + 1}'),
78            trailing: Icon(Icons.more_horiz),
79          ),
80        ),
81      );
82    }
83  }
```

Adding Separator – Approach 2, wrapped with Card




```dart
class _ChatsTabView extends StatelessWidget {
  final List items;

  _ChatsTabView(this.items);

  @override
  Widget build(BuildContext context) {
    return ListView.builder(
      itemCount: items.length,
      itemBuilder: (context, index) => Card(
        child: ListTile(
          leading: Icon(Icons.chat),
          title: Text(items[index]),
          subtitle: Text('Subtitle ${index + 1}'),
          trailing: Icon(Icons.more_horiz),
        ),
      ),
    );
  }
}
```

Adding Separator – Approach 3, ListTile.divideTiles()

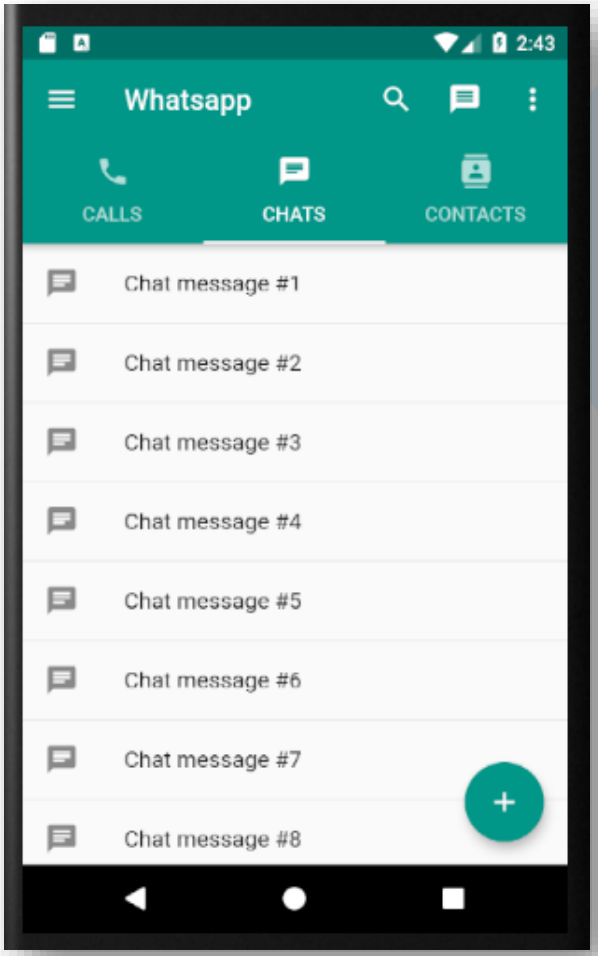


For static list, used with ListView() constructor

```dart
class _ChatsTabView extends StatelessWidget {
  final List items;

  _ChatsTabView(this.items);

  @override
  Widget build(BuildContext context) {
    return ListView(
      children: ListTile.divideTiles(
        context: context,
        tiles: [
          for (var item in items)    The for, if, and spread elements wer
            ListTile(
              leading: Icon(Icons.chat),
              title: Text(item),
            ),
        ],
      ).toList(),
    );
  }
}
```

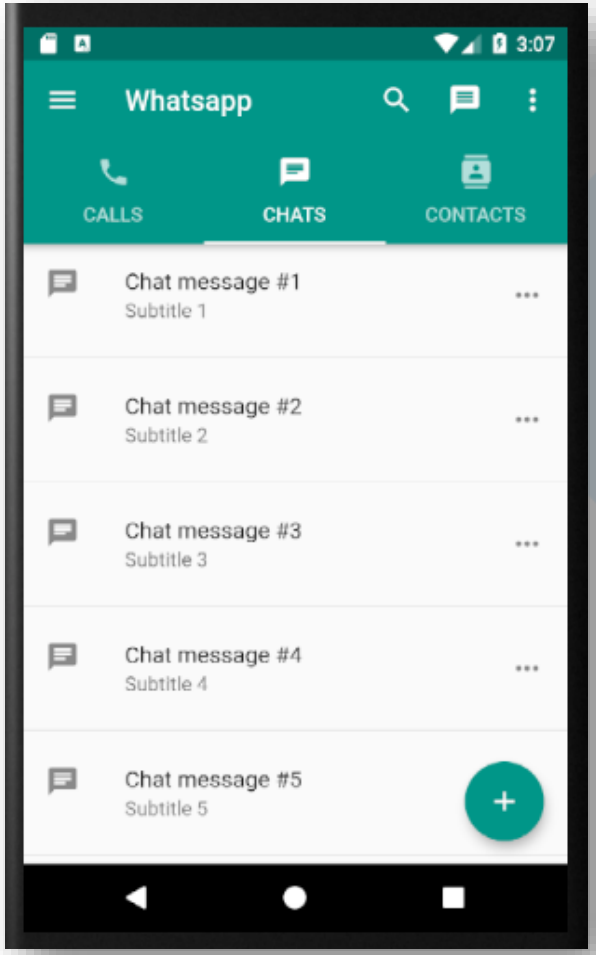
Adding Separator – Approach 3, ListTile.divideTiles()

This example is the same as in the previous slide, except this time high order method **map** is used instead of collection for




```dart
class _ChatsTabView extends StatelessWidget {
  final List items;

  _ChatsTabView(this.items);

  @override
  Widget build(BuildContext context) {
    return ListView(
      children: ListTile.divideTiles(
        context: context,
        tiles: items.map<Widget>(
          (item) => ListTile(
            leading: Icon(Icons.chat),
            title: Text(item),
          ),
        ),
      ).toList(),
    );
  }
}
```

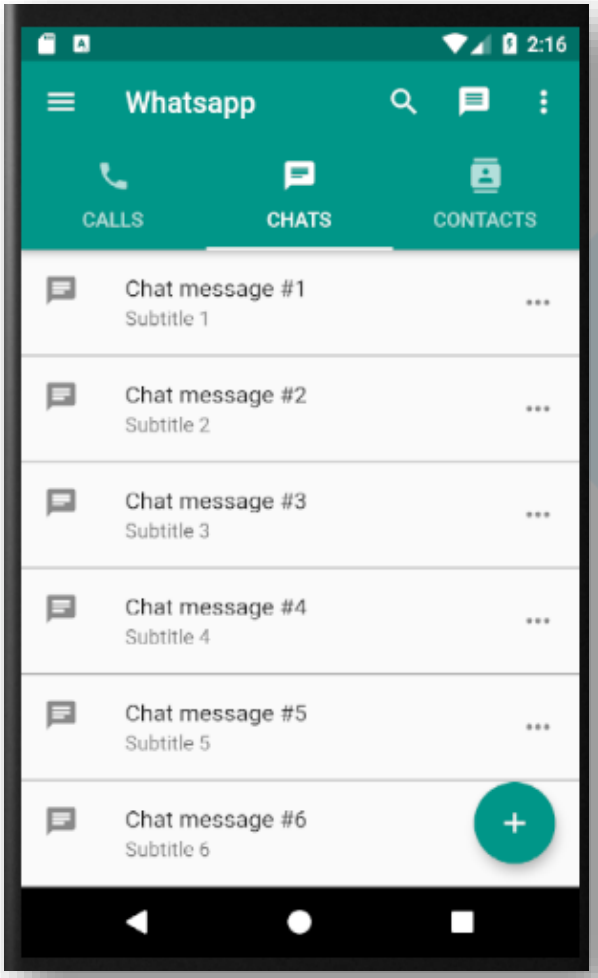
Adding Separator – Approach 4, ListView.separated()



Has dedicated builder (callback) to build separator

```dart
class _ChatsTabView extends StatelessWidget {
  final List items;

  _ChatsTabView(this.items);

  @override
  Widget build(BuildContext context) {
    return ListView.separated(
      itemCount: items.length,
      separatorBuilder: (context, index) => Divider(),
      itemBuilder: (context, index) => ListTile(
        leading: Icon(Icons.chat),
        title: Text(items[index]),
        subtitle: Text('Subtitle ${index + 1}'),
        trailing: Icon(Icons.more_horiz),
        ),
    );
  }
}
```

Adding SnackBar



- In this example, a snakebar will pop up when a chat message is long pressed
- Snackbar is called from a Scaffold

```
59  class _ChatsTabView extends StatelessWidget {
60    final List items;
61
62    _ChatsTabView(this.items);
63
64    @override
65    Widget build(BuildContext context) {
66      return ListView.separated(
67        itemCount: items.length,
68        separatorBuilder: (context, index) => Divider(),
69        itemBuilder: (context, index) => ListTile(
70          leading: Icon(Icons.chat),
71          title: Text(items[index]),
72          subtitle: Text('Subtitle ${index + 1}'),
73          trailing: Icon(Icons.more_horiz),
74          onLongPress: () => Scaffold.of(context).showSnackBar(SnackBar(
75            duration: Duration(seconds: 5),
76            backgroundColor: Theme.of(context).primaryColor.withOpacity(0.5),
77            content: Text('Item ${items[index]} has been long pressed'),
78          )),
79        ),
80      );
81    }
82  }
```

Drawer

- Adding drawers: drawer and endDrawer
- Open drawer programmatically

Adding Drawer (1)

- Drawer is part of Scaffold.
- Drawer is specified with the attributes **`drawer`** and **`endDrawer`**.
- To add the drawer menu on AppBar automatically, do not specify the attribute `leading` and `actions`.
- If `leading` and `actions` are specified manually, the drawers will need to ben opened programmatically.
- In the next example we are going to add a drawer that will open without needing explicit code.

Add to Scaffold, in class `Home`

```
12  class Home extends StatelessWidget {
13    @override
14    Widget build(BuildContext context) {
15      return DefaultTabController(
16        length: 3,
17        initialIndex: 1,
18        child: Scaffold(
19          appBar: _buildAppBar(),
20          body: _buildTabBarView(),
21          floatingActionButton: _buildFloatingActionButton(),
22          drawer: _buildDrawer(),
23          endDrawer: _buildEndDrawer(),
24        ),
25      );
26    }
27  } // class Home
```

Adding Drawer (2)

By not specifying the `leading` attribute, it means

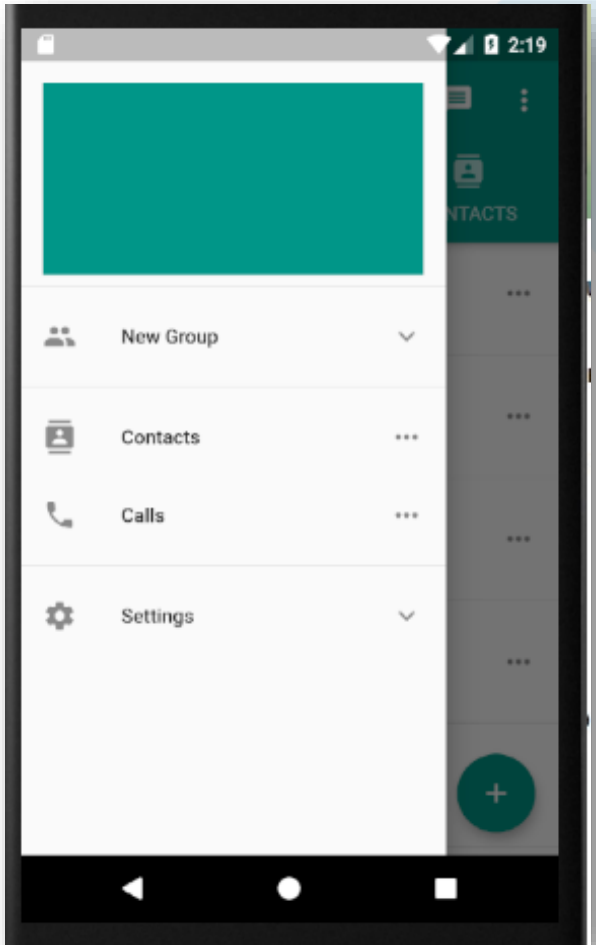
- Icon button for the drawer on the AppBar will be added automatically
- No explicit code for opening the drawer. Default code will be used

Unset the `leading` attribute of `AppBar`

```
35  AppBar _buildAppBar() {
36    return AppBar(
37      // leading: IconButton(
38      //   icon: Icon(Icons.menu),
39      //   onPressed: _doNothing,
40      // ),
41      title: Text('Whatsapp'),
42    > actions: <Widget>[ ···
47    > bottom: TabBar(tabs: [ ···
52    );
53  }
```

Adding Drawer (3)

- Drawer can be built with any widget.
- Common use case, using `ListView` widget with `DrawerHeader` on top followed by `ListTiles`



Build the drawer with Drawer widget

```
29  Drawer _buildDrawer() {
30    return Drawer(
31      child: ListView(
32        children: <Widget>[
33          DrawerHeader(
34            child: Container(color: Colors.teal),
35            ),
36          ListTile(
37            leading: Icon(Icons.group),
38            title: Text('New Group'),
39            trailing: Icon(Icons.expand_more),
40            ),
41          Divider(),
42          ListTile(
43            leading: Icon(Icons.contacts),
44            title: Text('Contacts'),
45            trailing: Icon(Icons.more_horiz),
46            ),
47          ListTile(
48            leading: Icon(Icons.call),
49            title: Text('Calls'),
50            trailing: Icon(Icons.more_horiz),
51            ),
52          Divider(),
53          ListTile(
54            leading: Icon(Icons.settings),
55            title: Text('Settings'),
56            trailing: Icon(Icons.expand_more),
57            ),
58        ],
59      ),
60    );
61  }
```

Open Drawer Programmatically (1)

By specifying the `leading` or `actions` attribute of `AppBar` , it means

- Custom Icon button will b e used for the drawer.
- The drawer will need to be open programmatically.

In the next example, we are going to open the `endDrawer` programmatically

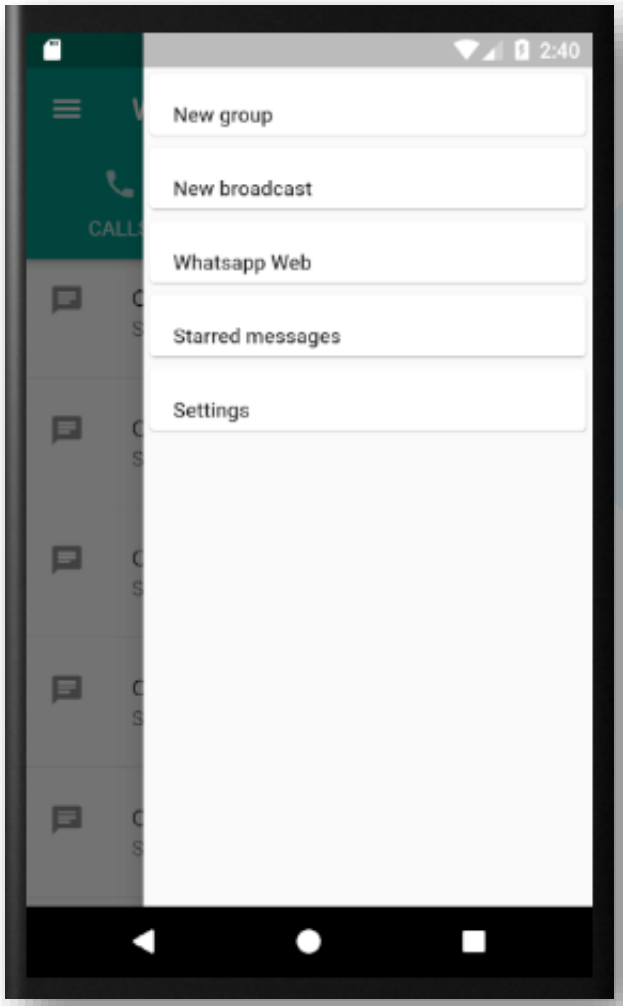
Note that drawers are under Scaffold, thus we need to open the drawer from Scaffold

Specify a callback for onTap of the last action button of `AppBar`

```
84  AppBar _buildAppBar() {
85    return AppBar(
86      // leading: IconButton(
87      //   icon: Icon(Icons.menu),
88      //   onPressed: _doNothing,
89      // ),
90      title: Text('Whatsapp'),
91      actions: <Widget>[
92        IconButton(icon: Icon(Icons.search), onPressed: _doNothing),
93        IconButton(icon: Icon(Icons.message), onPressed: _doNothing),
94        _AppBarActionMoreButton(),
95      ],
96      bottom: TabBar(tabs: [
97        Tab(text: 'CALLS', icon: Icon(Icons.call)),
98        Tab(text: 'CHATS', icon: Icon(Icons.chat)),
99        Tab(text: 'CONTACTS', icon: Icon(Icons.contacts)),
100     ]),
101   );
102 }
```

```
104 class _AppBarActionMoreButton extends StatelessWidget {
105   @override
106   Widget build(BuildContext context) {
107     return IconButton(
108       icon: Icon(Icons.more_vert),
109       onPressed: () => Scaffold.of(context).openEndDrawer());
110   }
111 }
```

Open Drawer Programmatically (2)



Build the drawer with Drawer widget

```
63  Drawer _buildEndDrawer() {
64    const menu = const <String>[
65      'New group',
66      'New broadcast',
67      'Whatsapp Web',
68      'Starred messages',
69      'Settings'
70    ];
71
72    return Drawer(
73      child: ListView.builder(
74        itemExtent: 50,
75        itemCount: menu.length,
76        itemBuilder: (context, index) =>
77            Card(child: ListTile(title: Text(menu[index]))),
78      ),
79    );
80  }
```

```
12  class Home extends StatelessWidget {
13    @override
14    Widget build(BuildContext context) {
15      return DefaultTabController(
16        length: 3,
17        initialIndex: 1,
18        child: Scaffold(
19          appBar: _buildAppBar(),
20          body: _buildTabBarView(),
21          floatingActionButton: _buildFloatingActionButton(),
22          drawer: _buildDrawer(),
23          endDrawer: _buildEndDrawer(),
24        ),
25      );
26    }
27  } // class Home
```

Summary

- Different ways to create flutter projects
- Introduction to Widgets
- Common widgets
- Refactoring