Lập trình đa nền tảng

Nguyễn Duy Nhật Viễn

Chương 5

Cơ bản Flutter

Cơ bản Flutter

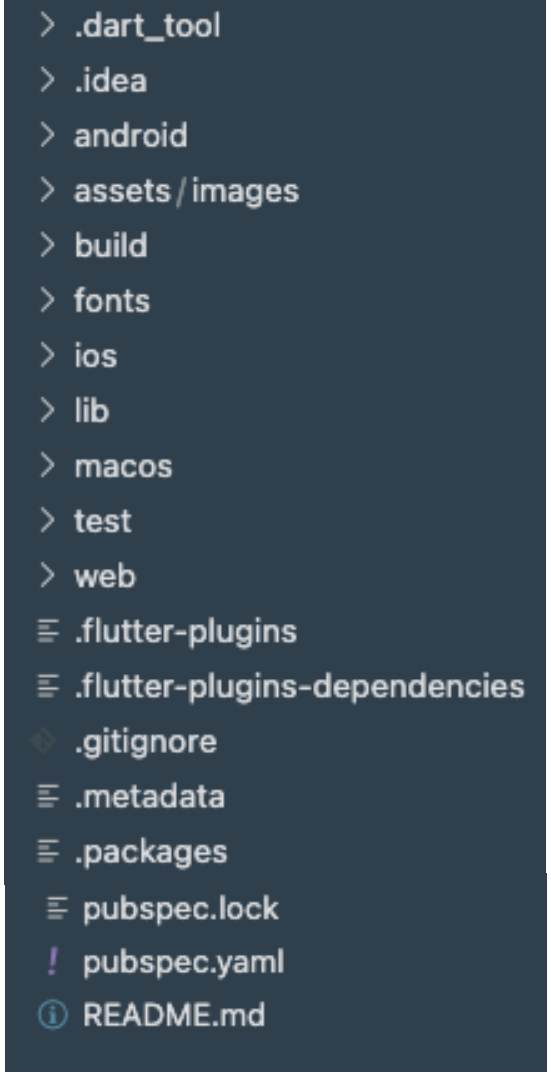
NGUYỄN DUY NHẬT VIỄN

Cấu trúc và công cụ

- Android Studio (AS)
- Visual Studio Code (VS Code)
- Emacs
- …

Cấu trúc thư mục và file trong 1 dự án Flutter

- Tùy theo ứng dụng viết cho nền tảng nào mà cấu trúc file và thư mục có khác nhau. Nhưng luôn tồn tại các file và thư mục chính sau:
- android / và ios /: Chứa các mã cho nền tảng cụ thể mà Flutter quản lý. Cấu trúc giống với các dự án Android trên Android Studio hoặc iOS trên XCode.
- lib /: Chứa mã nguồn Dart của ứng dụng.
- test /: chứa các kiểm tra đơn vị, widget và tích hợp.
- pubspec.yaml: Định nghĩa đóng gói Dart và liệt kê các phần phụ thuộc của ứng dụng Flutter.
- File .gitignore và README.md được sử dung với git, github (không liên quan đến source dự án).
- File .metadata và .packages là những file config, Flutter sử dụng để cấu hình dự án).



Cấu trúc thư mục

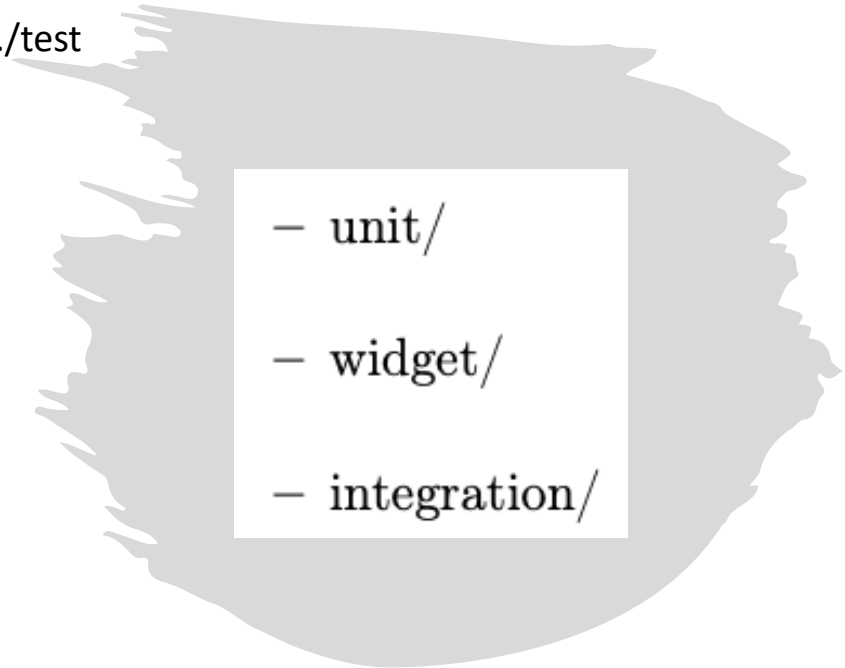
- Tổ chức các file và thư mục con trong thư mục ./lib
    - *Localizations: cho ứng dụng đa ngôn ngữ*

- routes/
- models/
- widgets/
- main.dart
- routes.dart

- localizations/
- routes/
- widgets/
- models/
    * blocs/
    * providers/
    * repositories/
    * ...
- main.dart
- routes.dart

Cấu trúc thư mục

- Tổ chức các file và thư mục con trong thư mục ./test

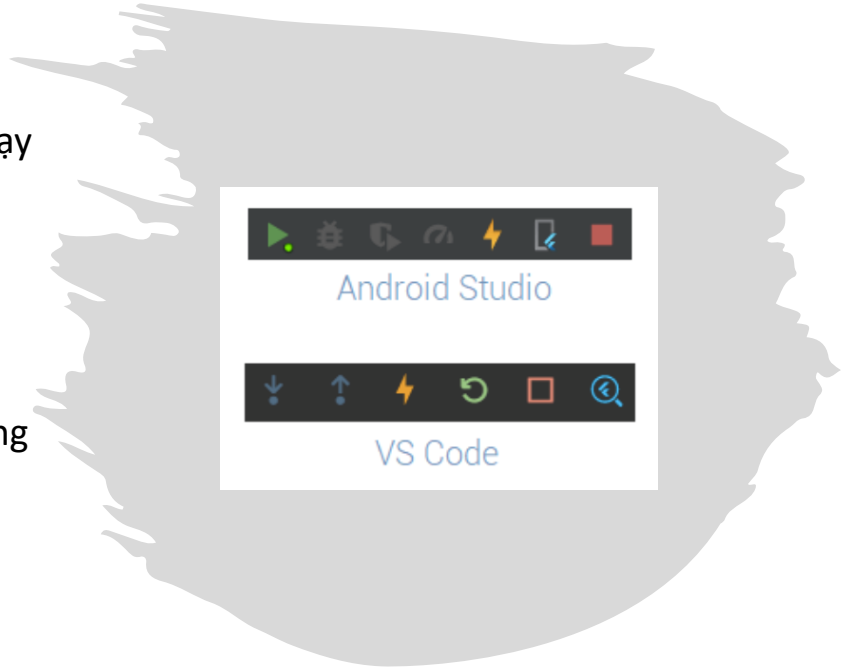


File pubspec.yaml

- Cung cấp cho quyền kiểm soát: phiên bản ứng dụng, các phụ thuộc (dependencies) được sử dụng bởi Flutter, resources/assets của ứng dụng.
- Một số thông tin:
  - Phiên bản các gói phụ thuộc, ví dụ: cupertino_icons: ^1.0.2
  - Sdk: Yêu cầu SDK của môi trường, ví dụ: environment: sdk: ">=2.7.0 <3.0.0"
  - uses-material-design: cho phép hay không việc sử dụng các icon của Google material design
  - Dependencies: khai báo các gói phụ thuộc (được sử dụng) trong dự án. Các gói này có thể tìm thấy trong https://pub.dev.
  - Assets: chỉ định các đường dẫn đến tài nguyên tĩnh mà ứng dụng sẽ sử dụng như hình ảnh, vectơ SVG, file âm thanh / video hoặc văn bản đơn giản.
  - Fonts: Khai báo các font trong dự án. Có thể vào https://fonts.google.com/ để tải các font và import vào dự án

Hot Reload

- Làm tươi giao diện người sử dụng UI (user interface) bằng phím nóng
- Tuy nhiên, một số trường hợp, phải dừng và chạy lại ứng dụng hoàn toàn:
  - khi thực hiện các thay đổi đối với phương thức initState ()
  - khi thay đổi định nghĩa của một lớp thành một enum và ngược lại,
  - khi thực hiện các thay đổi đối với các trường static trong các lớp,
  - khi thực hiện các thay đổi đối với mã bên trong void main () {}.



Cấu trúc chương trình Flutter

main.dart ●

lib > main.dart > ...

```dart
// Bước 1: import thư viện material - cung cấp các widget để code
import 'package:flutter/material.dart';

// Bước 2: khai báo hàm main - là nơi mà code sẽ thực thi đầu tiên
Run | Debug
void main() {
  // Bước 3: gọi hàm runApp truyền vào 1 object MyApp
  // MyApp chính là widget root, là gốc của 1 cây widget sau này
  runApp(MyApp());
}

//Bước 4: Định nghĩa lớp MyApp
class MyApp extends StatelessWidget {
  // Widget này là root của ứng dụng
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo', //title của ứng dụng
      theme: ThemeData(
        // Theme của ứng dụng
        primarySwatch: Colors.blue, //màu toolbar
        visualDensity: VisualDensity.adaptivePlatformDensity,
        //mật độ hình ảnh thích ứng với nền tảng
      ), // ThemeData
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    ); // MaterialApp
  }
}
```

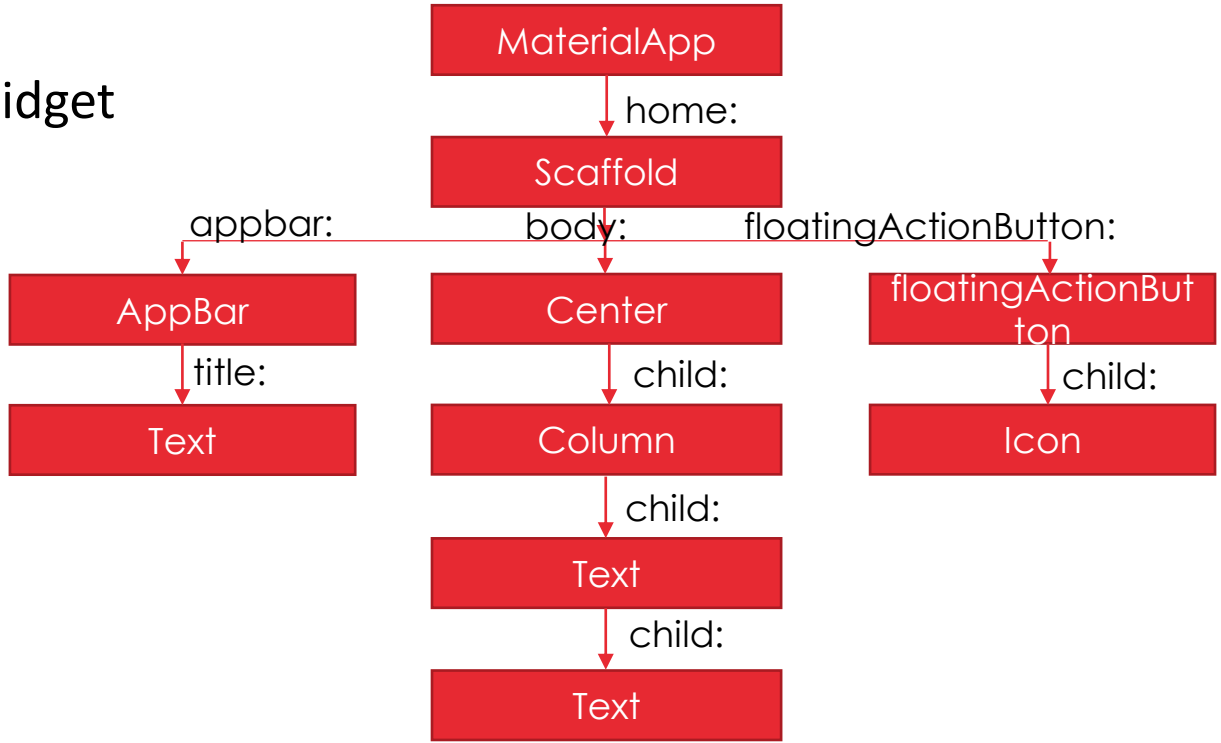
Cấu trúc chương trình Flutter

```dart
class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);
  // Widget là trang chủ của ứng dụng. Stateful nghĩa là có đối tượng State
  // (định nghĩa bên dưới) chứa các trường ảnh hưởng đến giao diện
  // Các trường trong lớp con wiged luôn được định nghĩa là "final"
  final String title;
  @override
  _MyHomePageState createState() => _MyHomePageState();
}
```

```dart
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;
  void _incrementCounter() {
    setState(() { // setState() để cập nhật giá trị
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    // Phương thức này được chạy lại mỗi khi setState() được gọi

    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ), // AppBar
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'You have pushed the button this many times:',
            ), // Text
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.headline4,
            ), // Text
          ], // <Widget>[]
        ), // Column
      ), // Center
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: Icon(Icons.add),
      ), // FloatingActionButton
    ); // Scaffold
  }
}
```

Cấu trúc chương trình Flutter

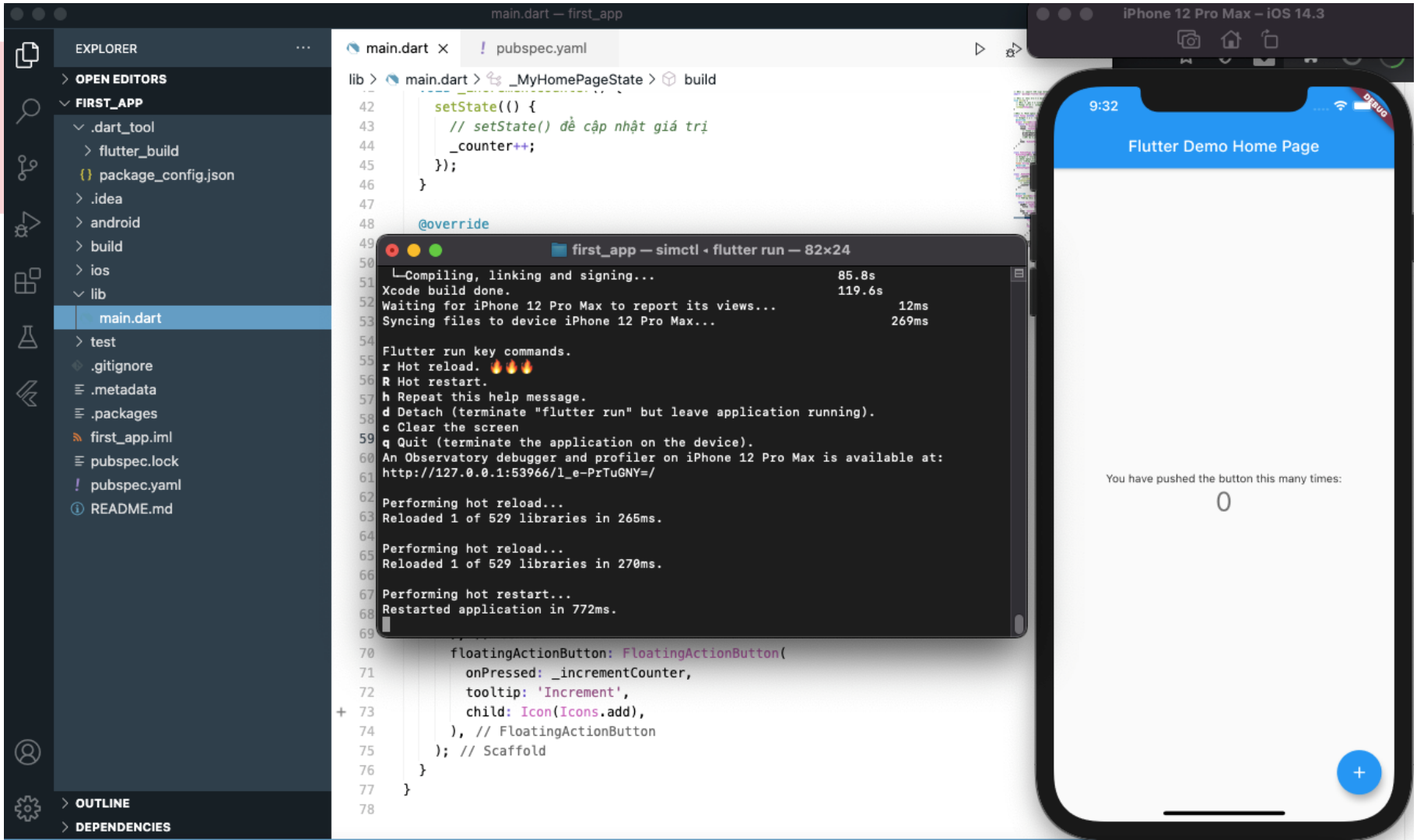
- Cây widget



File **pubspec.yaml**

pubspec.yaml

```yaml
# lưu ý: chú giải ở file này bằng dấu #, không phải //
name: first_app # tên project
description: A new Flutter project. # phần mô tả project

publish_to: 'none' # bỏ dòng này nếu muốn publish lên pub.dev

version: 1.0.0+1 # version của project

environment: # khai báo môi trường
  sdk: ">=2.7.0 <3.0.0" #sdk của flutter

dependencies: #khai báo thư viện
  flutter:
    sdk: flutter
  # các thư viện dùng trong chương trình được khai báo ở đây
  cupertino_icons: ^1.0.0 #sử dụng cupertino icons của ios

dev_dependencies:
  flutter_test:
    sdk: flutter

flutter:

  uses-material-design: true # cho phép sử dụng Material icons

  # để thêm các assets ứng dụng, sử dụng cú pháp
  # assets:
  #   - images/a_dot_burr.jpeg
  #   - images/a_dot_ham.jpeg

  # thêm font, sử dụng
  # 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
```

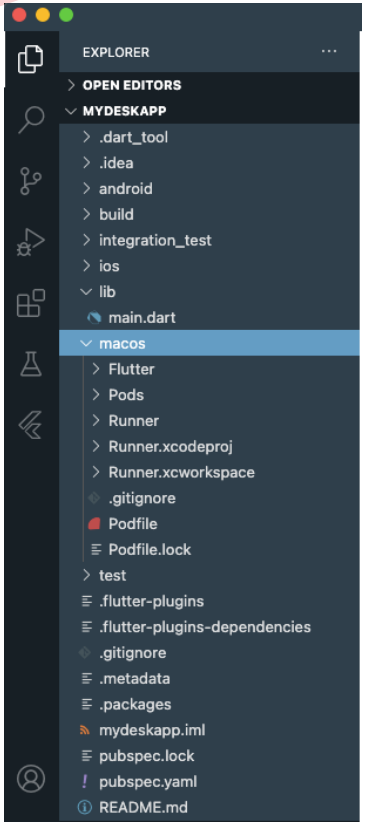

main.dart — first_app
EXPLORER
OPEN EDITORS
FIRST_APP
.dart_tool
flutter_build
package_config.json
.idea
android
build
ios
lib
main.dart
test
.gitignore
.metadata
.packages
first_app.iml
pubspec.lock
pubspec.yaml
README.md
OUTLINE
DEPENDENCIES
main.dart
pubspec.yaml
lib > main.dart > _MyHomePageState > build
setState(() {
// setState() để cập nhật giá trị
_counter++;
});
}
@override
first_app — simctl ◂ flutter run — 82×24
└Compiling, linking and signing... 85.8s
Xcode build done. 119.6s
Waiting for iPhone 12 Pro Max to report its views... 12ms
Syncing files to device iPhone 12 Pro Max... 269ms
Flutter run key commands.
r Hot reload. 🔥🔥🔥
R Hot restart.
h Repeat this help message.
d Detach (terminate "flutter run" but leave application running).
c Clear the screen
q Quit (terminate the application on the device).
An Observatory debugger and profiler on iPhone 12 Pro Max is available at:
http://127.0.0.1:53966/l_e-PrTuGNY=/
Performing hot reload...
Reloaded 1 of 529 libraries in 265ms.
Performing hot reload...
Reloaded 1 of 529 libraries in 270ms.
Performing hot restart...
Restarted application in 772ms.
floatingActionButton: FloatingActionButton(
onPressed: _incrementCounter,
tooltip: 'Increment',
child: Icon(Icons.add),
), // FloatingActionButton
); // Scaffold
}
}
iPhone 12 Pro Max – iOS 14.3
9:32
Flutter Demo Home Page
You have pushed the button this many times:
0

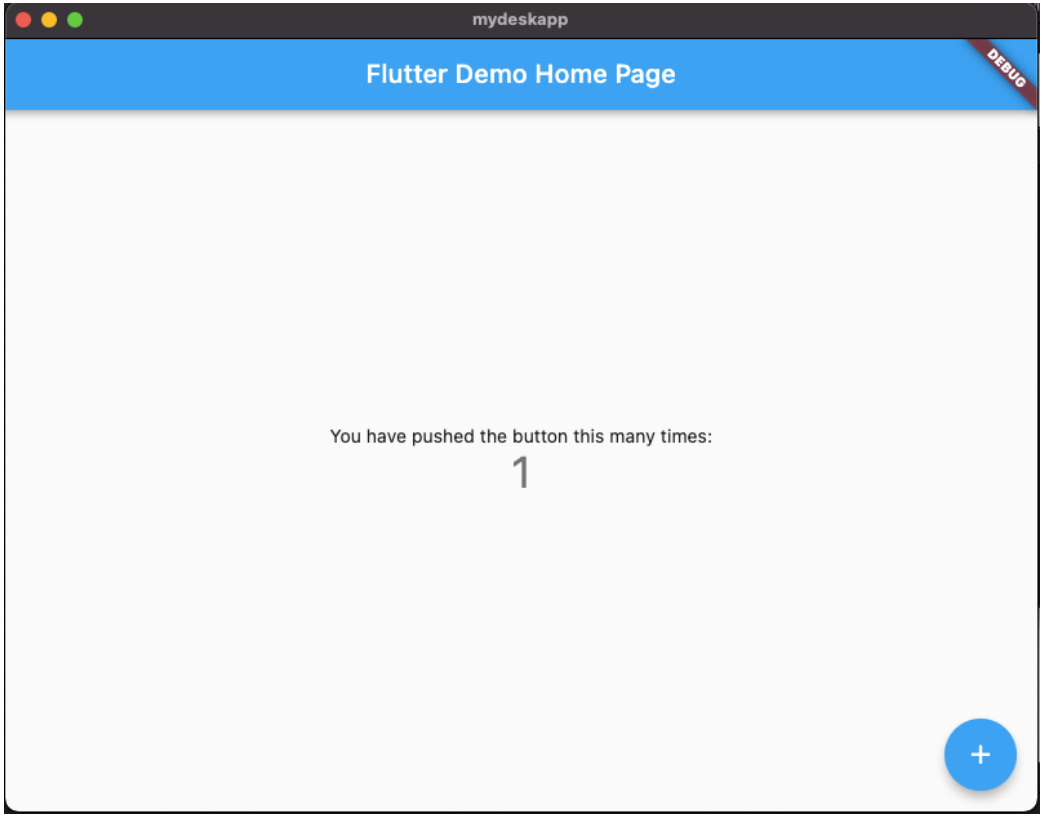
Một số câu lệnh run và build app

- Run app android hoặc ios:
  - flutter devices
  - flutter run
- Đối với desktop app, phải cấu hình trước khi tạo app:
  - flutter channel dev
  - flutter upgrade
  - flutter config --enable-<platform>-desktop
- Platform là windows, macos hoặc linux:
  - flutter config --enable-windows-desktop
  - flutter config --enable-macos-desktop
  - flutter config --enable-linux-desktop

- flutter run -d windows
- flutter run -d macos
- flutter run -d linux

Desktop app cho macos

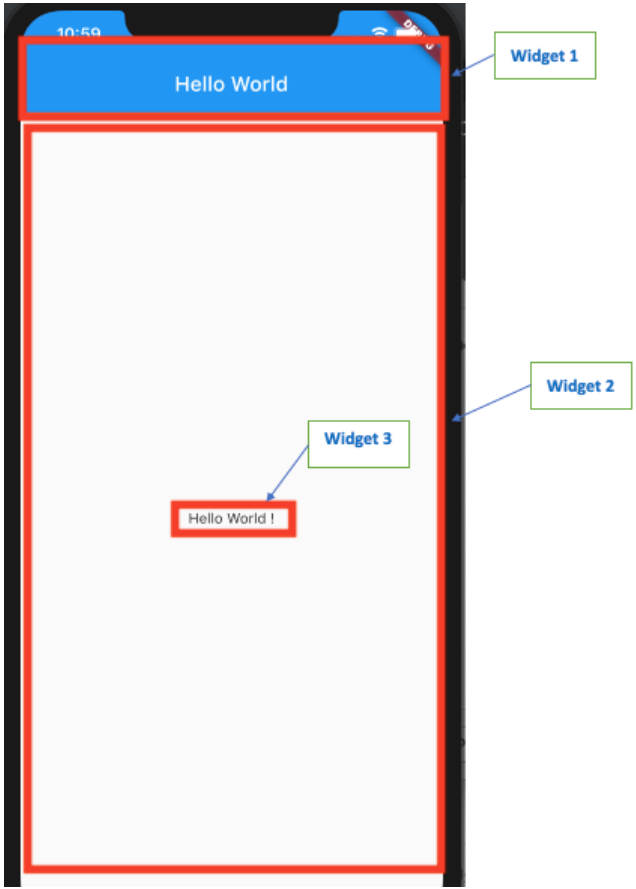




Widgets và State

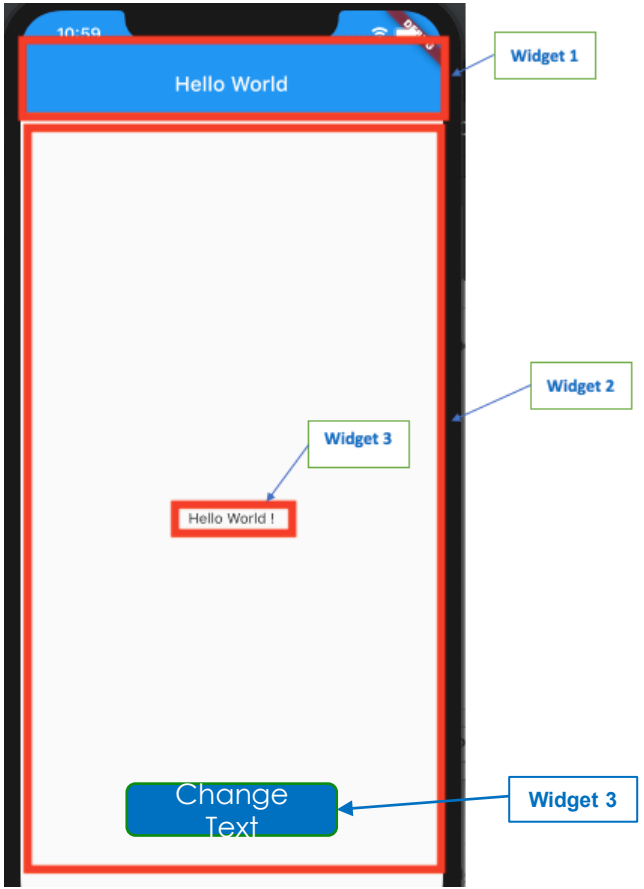
Khái niệm

- Trong Flutter, tất cả những gì xuất hiện lên màn hình đều gọi là Widget (phụ tùng).
- Các thành phần - màn hình trong ứng dụng Flutter được tạo bởi các Widget liên kết lại với nhau.
- Widget là các “đối tượng cấu hình” dùng để tạo nên ứng dụng Flutter. Vậy, việc tạo ứng dụng Flutter là tạo ra các Widget và liên kết chúng lại với nhau.



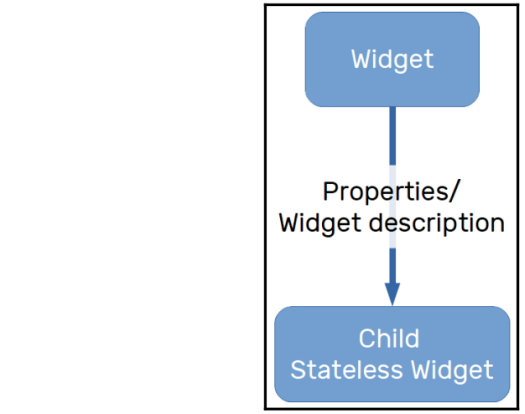
Khái niệm

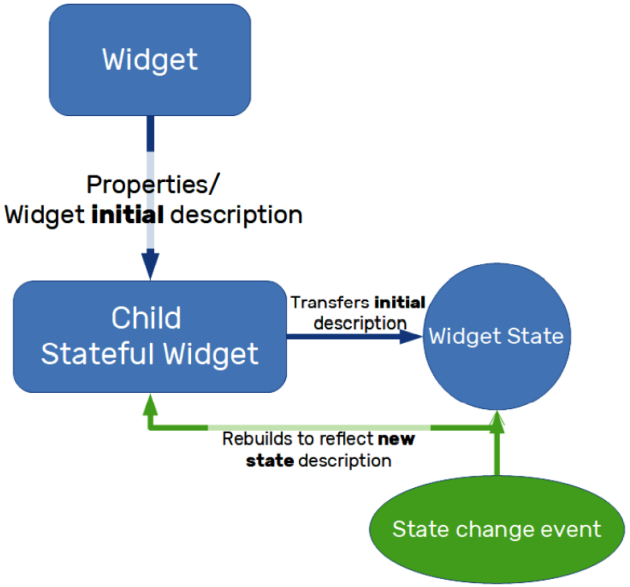
- State là dữ liệu hoặc thông tin được lưu trong bộ nhớ.
- State có thể kích hoạt quá trình **rebuild** giao diện người dùng hoặc các phần nhất định của giao diện người dùng dựa trên dữ liệu đã thay đổi.
- Ví dụ: Ứng dụng như hình bên. Mỗi lần bấm nút Change Text thì dòng text trên màn hình thay đổi. Ta nói State của Text thay đổi hay State của Ứng dụng đã thay đổi mỗi khi nhấn nút.
- Hai loại State là:
  - StatelessWidget
  - StatefulWidget.



Khái niệm

- **StatelessWidget**
  - StatelessWidget là một Widget được định nghĩa trong Flutter SDK, Widget này chỉ nhận dữ liệu và hiển thị dữ liệu thụ động không liên quan đến state.
  - StatelessWidget: Widget không có State hay Widget không trạng thái.
- **StatefulWidget**
  - StatefulWidget là một Widget được định nghĩa trong Flutter SDK.
  - StatefulWidget: Widge có State hay có trạng thái.





StatelessWidget

- Các Widget kế thừa StatelessWidget class từ material package được coi là một stateless widget bởi Flutter.
- StatelessWidget chỉ quan tâm đến việc hiển thị một số dữ liệu nhất định với một style nhất định.
- Chỉ được tạo một lần duy nhất và không thay đổi khi người dùng tương tác với chúng, ngay cả khi dữ liệu bên trong chúng thay đổi trừ khi dữ liệu bên ngoài (Widget cha) cung cấp cho chúng thay đổi.
- Phương thức xây dựng của các widget này chỉ có thể được kích hoạt nếu widget cha của các widget này được rebuild hoặc dữ liệu được cung cấp cho chúng bên ngoài thông qua hàm dựng (constructor) của chúng.
- Nếu phương thức build(){...} của parent widget được kích hoạt thì child stateless widget cũng được rebuild.
- Nếu Provider được đính kèm với stateless widget và widget đó là consumer hoặc active listener đối với provider thì ngay sau khi các giá trị của provider thay đổi, stateless widget sẽ rebuild.

```
class MyApp extends statelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}
```

StatefulWidget

- Tất cả các widget kế thừa StatefulWidget class được xem là các stateful widget.
- StatefulWidget kích hoạt các phương thức build của chúng ngay khi dữ liệu bên trong thay đổi hoặc dữ liệu bên ngoài được cung cấp cho Widget thông qua hàm dựng.

```
class MyHomePage extends statefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);
  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}
```

StatefulWidget

- Các stateful widget không chỉ là một class mà là sự kết hợp của hai class.
  - Class thứ nhất kế thừa StatefulWidget và override phương thức createState(). Phương thức createState() được khai báo bởi StatefulWidget class.
  - Class thứ hai bao gồm tất cả logic liên quan đến widget state.

```
class MyHomePage extends statefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);
  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}
```

```
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(

        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'You have pushed the button this many times:',
            ),
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.display1,
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: Icon(Icons.add),
      ), // This trailing comma makes auto-formatting nicer.
    );
  }
}
```

StatefulWidget

- Phương thức @override để cho Flutter biết rằng ta đang trả về một đối tượng mới dựa trên class thứ hai và có thể kết nối cả hai class này với nhau.
- Class đầu tiên có thể được tạo lại ngay khi dữ liệu bên ngoài được cung cấp thông qua hàm dựng. Nhưng vì ta cần lưu state bên trong của Widget khi việc rebuild do thay đổi từ Widget cha. Do đó, cần hai class: 01 để kích hoạt build, nhận vào các dữ liệu từ bên ngoài và 01 để kích hoạt rebuild cho chính bản thân Widget đó.

```
class MyHomePage extends statefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);
  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}
```

```
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(

        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'You have pushed the button this many times:',
            ),
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.display1,
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: Icon(Icons.add),
      ), // This trailing comma makes auto-formatting nicer.
    );
  }
}
```

StatefulWidget

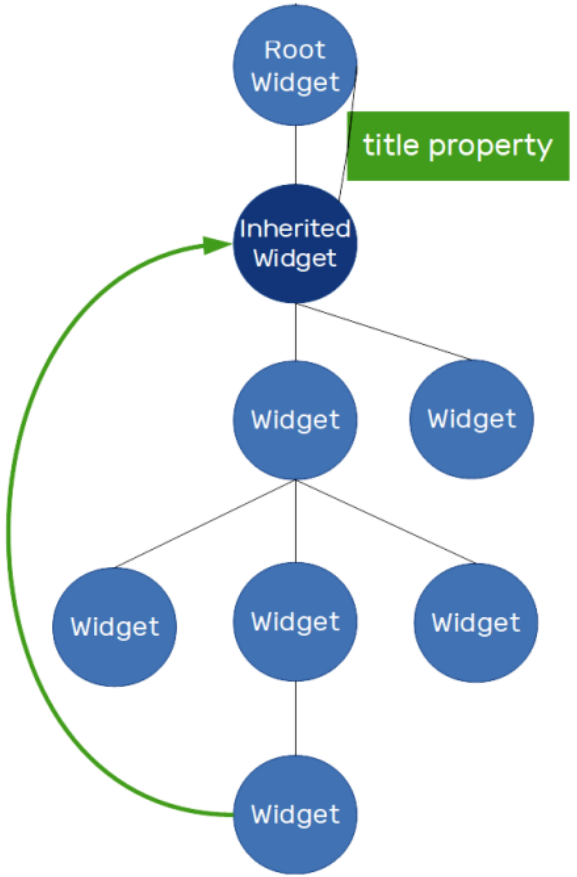
- Phương thức **setState**(){...} được sử dụng để thay đổi trạng thái của State, phương thức này giúp build lại State
- Trong ví dụ, hàm **_incrementCounter()** được thực thi khi bấm **FloatingActionButton**. Hàm này bọc tất cả logic/code bên trong để thay đổi dữ liệu. Dữ liệu nội bộ của hàm này này lại đang được sử dụng cho hàm build của widget. Vì vậy, ngay khi dữ liệu này thay đổi, hàm build sẽ được kích hoạt.

```
class MyHomePage extends statefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);
  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}
```

```
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(

        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'You have pushed the button this many times:',
            ),
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.display1,
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: Icon(Icons.add),
      ), // This trailing comma makes auto-formatting nicer.
    );
  }
}
```

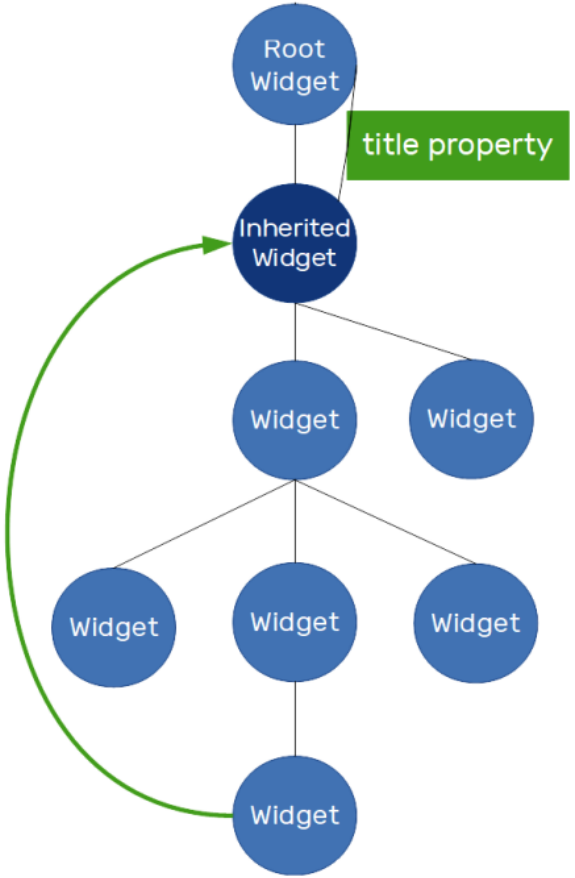
InheritedWidget

- InheritedWidget là nơi lưu trữ dữ liệu và cung cấp dữ liệu cho các widget con trong widget tree.
- Tất cả các widget con của InheritedWidget đều có thể truy cập vào InheritedWidget để lấy dữ liệu.
- Không cần thiết phải truyền dữ liệu xuống từng widget con từ InheritedWidget.



InheritedWidget

- Các widget con sử dụng phương thức inheritFromWidgetOfExactType(Inherited Widget) của Lớp BuildContext để nhận kiểu InheritedWidget làm tham số và sử dụng cây để tìm widget gốc của kiểu dữ liệu được yêu cầu.



Widget Tree, Element Tree và Render Tree

Widget Tree

- Widget Tree là tất cả các Widget đang sử dụng để xây dựng ứng dụng.
- Các widget có thể được khai báo lồng ghép với nhau để tạo nên bố cục mong muốn.
- Widget Tree được xây dựng bởi Flutter khi gọi phương thức build(){...} từ code, là một loạt các cài đặt cấu hình mà Flutter sẽ xử lý.
- Widget Tree không chỉ xuất hiện ra trên màn hình một lần mà sẽ cho Flutter biết những gì sẽ vẽ lên màn hình ở lần tiếp theo.
- Widget tree được rebuild rất thường xuyên.

Element Tree

- Element Tree liên kết vào Widget Tree, là thông tin được thiết lập cho các đối tượng/phần tử thực sự được hiển thị. Element Tree rất hiếm khi rebuild.
- Element Tree được quản lý theo một cách khác và sẽ không rebuild khi phương thức build(){...} được gọi.
- Flutter sẽ tự động tạo một element cho mỗi Widget trong Widget Tree ngay khi Flutter xử lý Widget ở lần đầu tiên.
- Element là một đối tượng được quản lý trong bộ nhớ bởi Flutter, liên quan đến Widget trong Widget Tree.
- Element chỉ giữ một tham chiếu tới Widget (trong Widget Tree) đang giữ các thông số giao diện đầu cuối .
- Khi có một stateful widget, Flutter sẽ tạo element và sau đó gọi phương thức createState() để tạo một state object mới dựa trên state class.
- Do đó, state là một đối tượng độc lập trong một Stateful Widget được kết nối với element trong Element Tree và Widget trong Widget Tree

Render Tree

- Render Tree đại diện của các element/đối tượng thực sự được hiển thị trên màn hình.
- Render Tree cũng không rebuild thường xuyên.
- Element Tree cũng được liên kết với Render Tree. Element trong Element Tree trỏ đến render object mà hiển thị trên màn hình.
- Khi Flutter thấy một element chưa được render trước đó thì sẽ tham chiếu đến Widget trong Widget Tree để thiết lập, sau đó tạo một element trong element tree.
- Flutter cũng có một giai đoạn layout, là giai đoạn tính toán và lấy không gian diện tích có sẵn trên màn hình, chiều, kích thước, hướng, ...v.v.
- Flutter cũng có một giai đoạn khác để thiết lập các listeners với các Widget để có thể thao tác các sự kiện, v.v.

Cách thực thi phương thức build()

- Phương thức build(){...} được Flutter gọi bất cứ khi nào state thay đổi.
- Về cơ bản, có hai kích hoạt quan trọng có thể dẫn đến việc rebuild.
  - Khi phương thức setState(){...} được gọi trong một Stateful Widget.
  - Khi MediaQuery hoặc lệnh Theme.of(...) được gọi, bàn phím ảo xuất hiện hoặc biến mất, v.v thay đổi dữ liệu sẽ tự động kích hoạt phương thức build(){...}.
- Việc gọi setState(){...} sẽ đánh dấu phần tử tương ứng là dirty. Đối với lần render tiếp theo, diễn ra 60 lần mỗi giây, Flutter sau đó sẽ xem xét đến các thông tin thiết lập mới được tạo bởi phương thức build(){...} và sau đó cập nhật màn hình.
- Tất cả các Widget lồng vào nhau bên trong Widget được đánh dấu là dirty sẽ được tạo các đối tượng mới widget/dart cho chúng. Do đó, một Widget Tree mới sẽ được tạo ra tương ứng các phiên bản mới của tất cả các Widget này.

Thuộc tính key của Widget

- Key là một định danh dùng để xác định danh tính cho các Widget, Element
- Trong hàm tạo/dựng của các lớp statelessWidget và statefulWidget có tham số key. Tham số này giúp kết xuất từ widget tree sang cây element tree.
- Thuộc tính key giúp duy trì trạng thái của widget con giữa các lần rebuild.
- Cách sử dụng key phổ biến nhất là khi ta xử lý các tập hợp các widget có cùng loại. Nếu không có key, element tree sẽ không biết trạng thái nào tương ứng với widget con nào, vì đều có cùng một kiểu dữ liệu.
- Nếu Widget được khai báo Key thì Element không chỉ so sánh Widget Type mà còn so sánh cả Key của widget cũ mà nó đang giữ tham chiếu và widget mới nữa.
  - Nếu cả Widget Type và Key đều giống nhau thì nó sẽ chỉ update bản thân nó bằng cách cho biến widget trỏ đến Widget mới.
  - Nếu cùng Widget Type nhưng Key khác nhau thì Element đó sẽ bị deactivate.
  - Nếu Widget Type khác nhau thì Element đó sẽ bị dispose

Các Widget tích hợp

Các Widget cơ bản

Scaffold

- Scaffold là một cấu trúc bố trí cơ bản dựa trên material design.
- Nếu sử dụng material design, màn hình ứng dụng sẽ có Scaffold làm cơ sở.
- Lớp Scaffold là một widget có khả năng mở rộng lấp đầy không gian sẵn có hoặc lấp đầy màn hình.
- Scaffold cung cấp API để hiển thị các widget chính của ứng dụng như Drawer, SnackBar, Bottom-Sheet, FloatingActionButton, AppBar, BottomNavigationBar,..

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('First App'),
    ), // AppBar
    body: Center(
        child:
        Text (
          'Hello World',
        ) // Text
    ), // Center
  ); // Scaffold
}
```

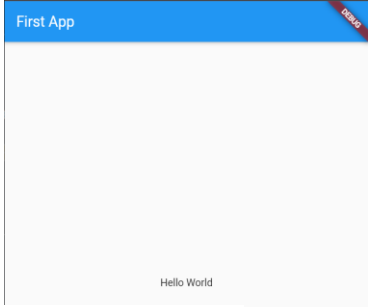
AppBar

- AppBar là một thanh ngang chủ yếu được hiển thị ở vị trí trên cùng của widget Scaffold.
- Nếu không khai báo có appBar, ta có một widget Scaffold không hoàn chỉnh.
- Scaffold sử dụng widget appBar có chứa các thuộc tính khác nhau như độ cao, tiêu đề, độ sáng, v.v.

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('First Flutter Application'),
    ), // AppBar
  ); // Scaffold
}
```

Body

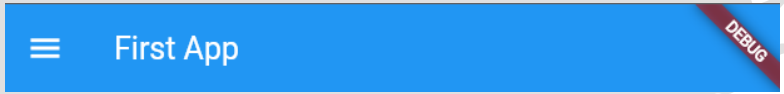
- Body là thuộc tính chính và bắt buộc khác của widget Scaffold.
- Nội dung chính trong Scaffold được hiển thị trong body.
- Vị trí của body ở dưới appBar và trên floatActionButton.
- Theo mặc định, các widget bên trong phần body được đặt từ vị trí phía trên, bên trái của không gian khả dụng.

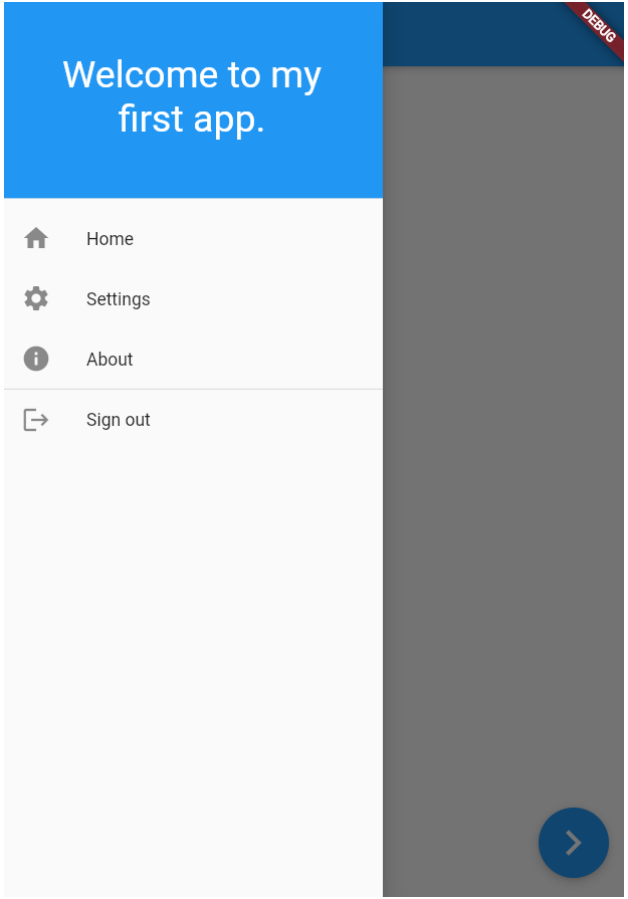



```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('First App'),
    ), // AppBar
    body: Center(
        child: Text(
      'Hello World',
    )), // Text // Center
    floatingActionButton: FloatingActionButton(
      onPressed: () {
        print('Next');
      },
      child: const Icon(Icons.arrow_forward_ios),
    ), // FloatingActionButton
  ); // Scaffold
}
```

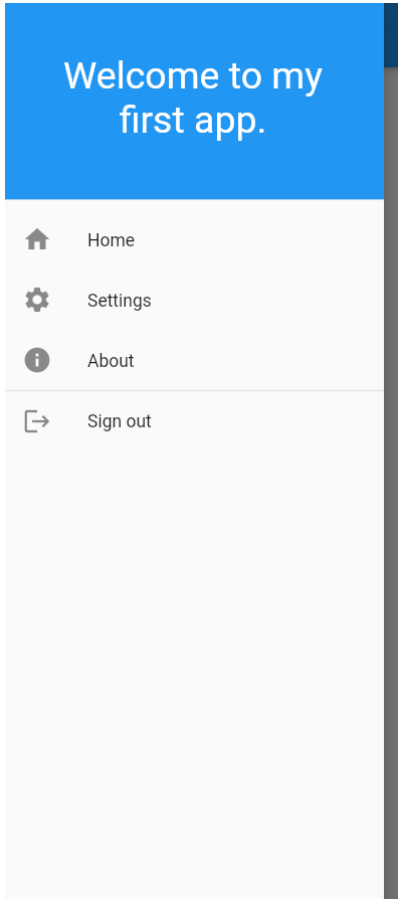
Drawer

- Drawer là một bảng điều khiển trượt được hiển thị ở bên cạnh của body.
- Thông drawer bị ẩn trên thiết bị di động, nhưng có thể vuốt drawer từ trái sang phải hoặc từ phải sang trái để truy cập menu drawer.
- Các cử chỉ cũng được thiết lập tự động để mở drawer.





```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('First App'),
    ), // AppBar
    body: Center(
        child: Text(
      'Hello World',
    )), // Text // Center
    floatingActionButton: FloatingActionButton(
      onPressed: () {
        print('Next');
      },
      child: const Icon(Icons.arrow_forward_ios),
    ), // FloatingActionButton
    drawer: Drawer(
      child: ListView(
        children: const <Widget>[
          DrawerHeader(
            decoration: BoxDecoration(
              color: Colors.blue,
            ), // BoxDecoration
            padding: EdgeInsets.fromLTRB(30, 45, 30, 45),
            child: Text(
              'Welcome to my first app.',
              textAlign: TextAlign.center,
              style: TextStyle(
                color: Colors.white,
                fontSize: 30,
              ), // TextStyle
            ), // Text
          ), // DrawerHeader
          ListTile(
            title: Text("Home"),
            leading: Icon(Icons.home),
          ), // ListTile
          ListTile(
            title: Text("Settings"),
            leading: Icon(Icons.settings),
          ), // ListTile
          ListTile(
            title: Text("About"),
            leading: Icon(Icons.info),
          ), // ListTile
          Divider(
            height: 0.2,
          ), // Divider
          ListTile(
            title: Text("Sign out"),
            leading: Icon(Icons.logout),
          ), // ListTile
        ], // <Widget>[]
      ), // ListView
    ), // Drawer
  ); // Scaffold
}
```



Text widget

- Text hiển thị xâu chuỗi của văn bản ở dạng

```
Text( "This is a text", )
```

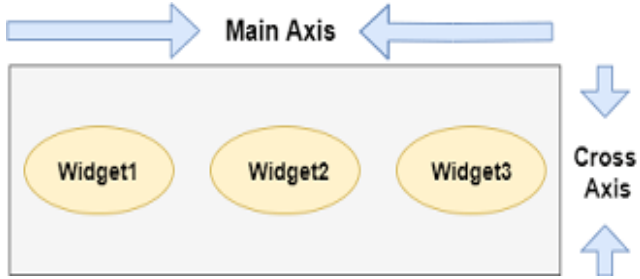
- Các đặc tính chung của Text widget như sau:
  - style: hiển thị các thuộc tính cho phép thay đổi màu chữ, màu nền, phông chữ, chiều cao dòng, kích thước phông chữ, v.v. Của văn bản
  - textAlign: kiểm soát căn chỉnh theo chiều ngang của văn bản, ví dụ căn giữa hoặc căn đều.
  - maxLines: chỉ định số dòng tối đa cho văn bản sẽ bị cắt bớt nếu vượt quá giới hạn.
  - overflow: xác định cách văn bản sẽ được cắt bớt khi bị tràn, cung cấp các tùy chọn như chỉ định giới hạn số dòng tối đa, thêm dấu chấm lửng ở cuối văn bản khi tràn.

Text widget

```
Text(
  'Hello world!',
  style: TextStyle(
    color: Colors.black,
    fontSize: 40,
    backgroundColor: Colors.white,
    fontWeight: FontWeight.bold,
  ), // TextStyle
), // Text
```

Hello world!

Column và Row



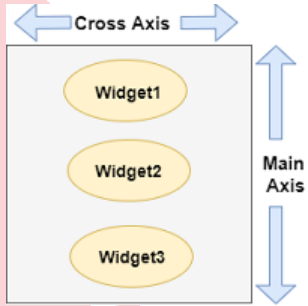
- Column và Rown là 2 widget giúp định hình 1 tập hợp các widget theo cột- chiều dọc (vertically) hoặc hàng - chiều ngang (horizontally).
  - Liên kết với các widget con bằng thuộc tính **children**, các widget con cách nhau bởi dấu phẩy.
  - Căn chỉnh các widget con bằng các thuộc tính: mainAxisAlignment, crossAxisAlignment.
- Khi sử dụng Row, các widget con được sắp xếp thành một hàng, theo chiều ngang. Vì vậy, trục chính của Row là nằm ngang.
- Sử dụng mainAxisAlignment trong Row cho phép bạn căn chỉnh các widget con của hàng theo chiều ngang (ví dụ: left, right).
- Trục chéo với trục chính của Row là trục dọc. Vì vậy, sử dụng crossAxisAlignment trong Row cho phép bạn xác định cách con của nó được căn chỉnh theo chiều dọc.

Column và Row

- Ta có thể căn chỉnh widget của row hay column với sự trợ giúp của các thuộc tính trong mainAxisAlignment và crossAxisAlignment:
  - **start:** Căng chỉnh từ điểm bắt đầu của trục chính.
  - **end:** Căng chỉnh cuối trục chính.
  - **center:** Căng chỉnh giữa trục chính.
  - **spaceBetween:** Căng chỉnh bằng các không gian trống giữa các con một cách đồng đều.
  - **spaceAround:** Căng chỉnh bằng các không gian trống giữa các con một cách đồng đều và một nửa không gian đó trước và sau widget con đầu tiên và cuối cùng.
  - **spaceEvenly:** Căng chỉnh bằng các không gian trống giữa các con một cách đồng đều và trước và sau widget con đầu tiên và cuối cùng.

Column và Row

- Khi sử dụng Column, các Widget con Column được sắp xếp theo chiều dọc, từ trên xuống dưới (theo mặc định). Vì vậy trục chính của nó là thẳng đứng.
- mainAxisAlignment trong một Column sẽ căn chỉnh các widget con của nó theo chiều dọc (ví dụ: top, bottom) và crossAxisAlignment xác định cách các con được căn chỉnh theo chiều ngang trong Column đó.




```
Column(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    Text(
      "Hello world!",
      style: TextStyle(
        backgroundColor: Colors.white,
        color: Colors.black,
        fontSize: 40,
        fontWeight: FontWeight.bold,
      ),
    ),
    Text(
      "This is sample app!",
      style: TextStyle(
        backgroundColor: Colors.white,
        color: Colors.grey,
        fontSize: 30,
        fontWeight: FontWeight.bold,
      ),
    ),
  ],
),
```

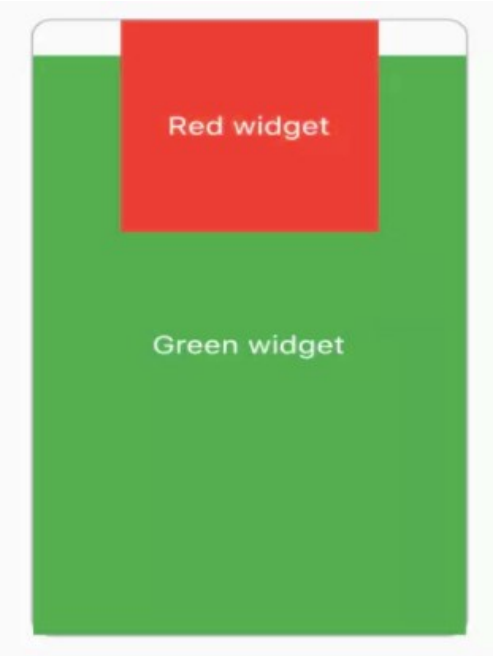
ListView

- ListView hoạt động tương tự như một Column hoặc Row với sự khác biệt duy nhất là các Widget con của nó có thể cuộn được.
- Có ba cách thực hiện hàm tạo cho ListView:
  - Mặc định lấy một list các widget trong thuộc tính con của nó. Đây là một lựa chọn tốt cho các list nhỏ vì để xây dựng nó, xử lý list sẽ xử lý mọi widget con.
  - ListView.builder sử dụng indexed builder để tạo các widget con yêu cầu. Lựa chọn này phù hợp với số lượng lớn widget con, vì mỗi khi xử lý list chỉ xử lý những widget con hiển thị.
  - ListView.custom lấy SliverChildDelegate, cho phép tùy chỉnh thêm nhiều khía cạnh cho ListView.

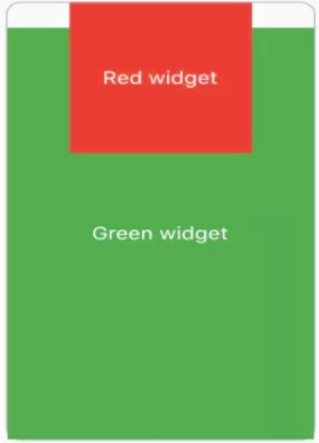
```
Widget _myListView(BuildContext context) {
    return ListView(
      children: <Widget>[
        ListTile(
          leading: Icon(Icons.wb_sunny),
          title: Text('Sun'),
        ), // ListTile
        ListTile(
          leading: Icon(Icons.brightness_3),
          title: Text('Moon'),
        ), // ListTile
        ListTile(
          leading: Icon(Icons.star),
          title: Text('Star'),
        ), // ListTile
      ], // <Widget>[]
    ); // ListView
  }
```

Stack

- Stack cũng là 1 tập hợp các widget, tuy nhiên, các widget con trong stack có thể chồng lên nhau.
- Thường đi kèm với widget này là widget Positioned để căn chỉnh vị trí của từng widget trong Stack



```dart
Container(
  width: 200,
  height: 350,
  decoration: BoxDecoration(
    border: Border.all(width: 1, color: Colors.black26),
    borderRadius: BorderRadius.circular(10),
  ), // BoxDecoration
  child: Stack(
    alignment: Alignment.center,
    children: [
      Positioned(
        top: 20,
        left: 0,
        right: 0,
        bottom: 0,
        child: Container(
          child: Center(
            child: Text(
              'Green widget',
              style: TextStyle(color: Colors.white),
            ), // Text
          ), // Center
          color: Colors.green,
        ), // Container
      ), // Positioned
      Positioned(
        top: 0,
        left: 40,
        right: 40,
        child: Container(
          child: Center(
            child: Text(
              'Red widget',
              style: TextStyle(color: Colors.white),
            )), // Text // Center
          height: 120,
          color: Colors.red,
        ), // Container
      ), // Positioned
    ],
  ), // Stack
), // Container
```



Container

- Đây là widget giống như một layout box, Container chứa các Widget bên trong, thuận tiện cho việc tạo cấu trúc trong màn hình ứng dụng.
- Container liên kết với widget con qua thuộc tính child.
- Các thuộc tính width, height giúp dễ dàng layout những view có kích thước đã xác định, đặc biệt thuộc tính decoration giúp trang trí cho view.

```
Container(
  width: 200,
  height: 200,
  decoration: BoxDecoration(
    color: Colors.amber,
    border: Border.all(width: 1, color: Colors.black26),
    borderRadius: BorderRadius.circular(10),
  ),
  padding: EdgeInsets.all(50),
  child: FlutterLogo(
    size: 50,
  ),
),
```

SizedBox

- Thường dùng để tạo các khoảng cách giữa các Widget hoặc dùng để tạo kích thước cho các Widget khác nhờ các thuộc tính kích thước như height, width.

```
// Tạo khoảng cách giữa TextField và Text
SizedBox(height: 20),

Text("Text Widget "),
// Tạo khoảng cách giữa Text và ElevatedButton
SizedBox(height: 20),

ElevatedButton(
  onPressed: () {
    print('You press button');
  },
  child: Text("Raise Button"),
), // ElevatedButton
```

SingleChildScrollView

- SingleChildScrollView widget giải quyết trường hợp 1 Column hoặc 1 Row có height hoặc width vượt qua độ lớn màn hình thiết bị.
- Chú ý: cần phải set giá trị cho thuộc tính scrollDirection của SingleChildScrollView (Axis.vertical là theo chiều dọc, Axis.horizontal theo chiều ngang).
- Nội dung chứa trong SingleChildScrollView sẽ được hiển thị khi vuốt hoặc kéo.

```
body: Center(
    child:SingleChildScrollView(
     child: ConstrainedBox(
       constraints: BoxConstraints(
         minHeight: 200,
       ), // BoxConstraints
       child: Column(
         mainAxisSize: MainAxisSize.min,
         mainAxisAlignment: MainAxisAlignment.spaceAround,
         children: <Widget>[
           Container(
             // A fixed-height child.
             color: const Color(0xffeeee00), // Yellow
             height: 120.0,
             alignment: Alignment.center,
             child: const Text('Fixed Height Content'),
           ), // Container
           Container(
             // Another fixed-height child.
             color: const Color(0xff008000), // Green
             height: 120.0,
             alignment: Alignment.center,
             child: const Text('Fixed Height Content'),
           ), // Container
         ], // <Widget>[]
       ), // Column
     ), // ConstrainedBox
    )), // SingleChildScrollView // Center
```

Expanded

- Thường được sử dụng để chia bố cục của widget cha thành các phần với tỉ lệ tương ứng
- Ví dụ: chia widget cha thành 3 widget con green, blue, grey theo tỉ lệ flex là 2:3:1 (mặc định flex = 1).

```
child: Column(
  children: [
    Expanded(
      flex: 2,
      child: Container(color: Colors.green),
    ), // Expanded
    Expanded(
      flex: 3,
      child: Container(color: Colors.blue),
    ), // Expanded
    Expanded(
      child: Container(color: Colors.grey),
    ), // Expanded
  ],
), // Column
```

Expanded

- Ngoài ra, Expanded còn được sử dụng trong trường hợp dynamic height hay width của widget con theo widget cha.
- Ví dụ: 1 Column chứa 2 Container trong đó 1 Container có chiều cao là 50, Container còn lại muốn có chiều cao bằng phần còn lại của Column, sử dụng Expanded để wrap Column thứ 2.

```
child: Column(
  children: [
    Container(
      width: double.infinity,
      height: 50,
      color: Colors.green,
    ), // Container
    Expanded(
      child: Container(color: Colors.grey),
    ), // Expanded
  ],
), // Column
```

Button

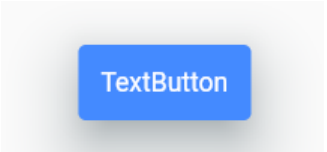
Button

- Flutter bổ sung 1 số button và bỏ đi 1 số button cũ với mục làm cho các button linh hoạt hơn và dễ cấu hình hơn thông qua các tham số hoặc chủ đề (theme) của phương thức khởi tạo.
- Các widget cũ: FlatButton, RaisedButton và OutlineButton được thay thế bằng TextButton, ElevatedButton và OutlinedButton. Mỗi lớp nút mới có chủ đề riêng: TextButtonTheme, ElevatedButtonTheme và OutlinedButtonTheme.
- Không tương thích ngược với widget cũ
- Ngoài ra còn có các nút sau:
- Nút nổi(Floating Button)
- Nút thả xuống(Drop Down Button)
- Nút biểu tượng(Icon Button)
- Nút Inkwell
- Nút PopupMenu

TextButton

- TextButton là một nút văn bản không có nhiều trang trí.
- Có hai thuộc tính bắt buộc là: child và onPressed().
- Chủ yếu được sử dụng trong các thanh công cụ hoặc hộp.
- Mặc định, TextButton không có màu nền và màu văn bản là màu đen. Tuy nhiên, ta có thể thiết lập màu thông qua style.

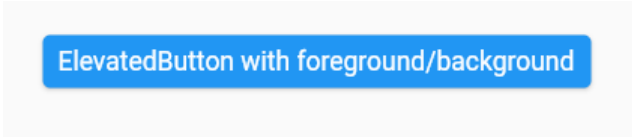
```
body: Center(
  child: TextButton(
    style: TextButton.styleFrom(
        primary: Colors.white,
        backgroundColor: Colors.blueAccent,
        shadowColor: Colors.blueGrey,
        elevation: 30,
        padding: EdgeInsets.all(20)),
    onPressed: () {},
    child: Text('TextButton'),
  ), // TextButton
), // Center
```



ElevatedButton

- ElevatedButton tương tự như TextButton, nhưng có độ cao(elevation) mặt định.
- Khi được nhấn, ElevatedButton tạo hiệu ứng đổi màu nền bằng màu chữ.
- Nút này có hai chức năng gọi lại.
  - onPressed (): được kích hoạt khi nhấn nút.
  - onLongPress (): được kích hoạt khi nhấn và giữ nút.
- Lưu ý: ElevatedButton ở trạng thái bị vô hiệu hóa nếu các lệnh gọi lại onPressed() và onLongPressed() không được chỉ định.

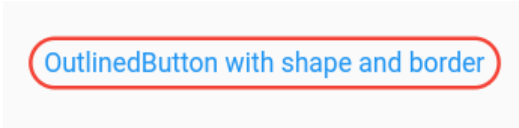
```
body: Center(
  child: ElevatedButton(
    style: ElevatedButton.styleFrom(
      primary: Colors.blue, // background
      onPrimary: Colors.white, // foreground
    ),
    onPressed: () {},
    child: Text('ElevatedButton with foreground/background'),
  ), // ElevatedButton
), // Center
```



OutlinedButton

- OutlinedButton tương tự như TextButton, nhưng có đường viền ngoài button thông qua 2 tham số là shape và size.

```
body: Center(
    child: OutlinedButton(
  style: OutlinedButton.styleFrom(
    shape: StadiumBorder(),
    side: BorderSide(width: 2, color: Colors.red),
  ),
  onPressed: () {},
  child: Text('OutlinedButton with shape and border'),
)), // OutlinedButton // Center
```



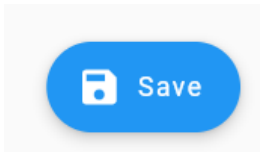
FloatingActionButton

- FloatingActionButton là một nút biểu tượng hình tròn kích hoạt hành động chính trong ứng dụng.
- Nên sử dụng nhiều nhất một nút FloatingActionButton trên mỗi màn hình.
- FloatingActionButton có hai loại:
  - FloatingActionButton: tạo một nút nổi hình tròn đơn giản với một widget con bên trong nó. Yêu cầu tham số child để hiển thị một widget.
  - FloatingActionButton.extended: tạo ra một nút nổi rộng cùng với một biểu tượng và nhãn bên trong nó. Thay vì một child, nút này sử dụng các nhãn và các thông số biểu tượng.

FloatingActionButton

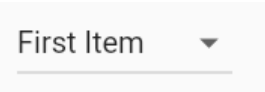
```
floatingActionButton: FloatingActionButton(
  child: Icon(Icons.navigation),
  backgroundColor: Colors.green,
  foregroundColor: Colors.white,
  onPressed: () => {},
), // FloatingActionButton
```

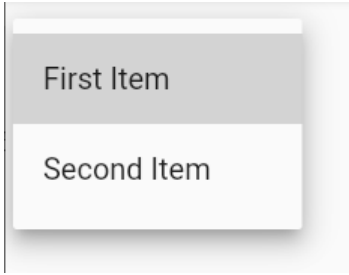
```
floatingActionButton: FloatingActionButton.extended(
  onPressed: () {},
  icon: Icon(Icons.save),
  label: Text("Save"),
), // FloatingActionButton.extended
```



DropdownButton

- Flutter cung cấp một widget DropdownButton để triển khai một danh sách thả xuống cho phép người dùng chọn bất kỳ mục nào từ nhiều tùy chọn.
- DropdownButton có thể được đặt ở mọi nơi trong ứng dụng.
- DropdownButton hiển thị mục hiện được chọn và một mũi tên mở menu để chọn một mục từ nhiều tùy chọn.






```
class _MyHomePageState extends State<MyHomePage> {
  int _value = 1;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
        body: Container(
      padding: EdgeInsets.all(20),
      child: DropdownButton(
          value: _value,
          items: [
            DropdownMenuItem(
              child: Text("First Item"),
              value: 1,
            ), // DropdownMenuItem
            DropdownMenuItem(
              child: Text("Second Item"),
              value: 2,
            ) // DropdownMenuItem
          ],
          onChanged: (int value) {
            setState(() {
              _value = value;
            });
          },
          hint: Text("Select item")), // DropdownButton
    )); // Container // Scaffold
  }
}
```

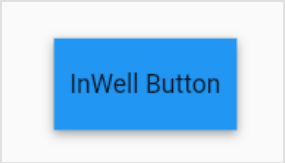
IconButton

- IconButton là một button với một biểu tượng (Icon), người dùng có thể click vào nó để thực hiện một hành động.
- IconButton sẽ không bao gồm nội dung văn bản, nếu bạn muốn có một button bao gồm biểu tượng và văn bản hãy sử dụng TextButton hoặc ElevatedButton.
- IconButton thường được sử dụng như một action trong property AppBar.actions, ngoaif ra, cũng được sử dụng trong nhiều tình huống khác.
- Vùng tương tác (hit region) của IconButton là vùng có thể cảm nhận được sự tương tác của người dùng, nó có kích thước nhỏ nhất là kMinInteractiveDimension (48.0) bất kể kích thước thực sự của Icon.

```
child: IconButton(
    icon: Icon(Icons.directions_bus),
    onPressed: () {
      print("Pressed");
    }), // IconButton
```

InkwellButton

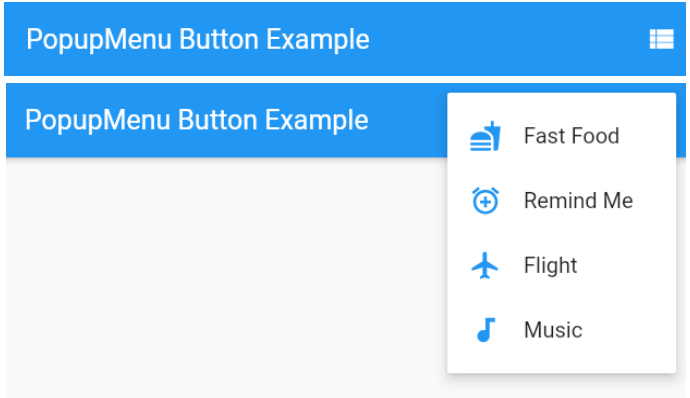
- InkwellButton là một khái niệm thiết kế material design, được sử dụng để phản hồi cảm ứng.
- InkwellButton tương tác bằng cách thêm phản hồi cử chỉ.
- Thường được sử dụng để thêm hiệu ứng gợn sóng.
- Lưu ý: Để có hiệu ứng gợn sóng, InkWell phải được bọc bởi Material.




```
child: Material(
  color: Colors.transparent,
  // color: Colors.white.withOpacity(0.0),
  child: InkWell(
    child: Center(
        child: Text("InWell Button",
            style: Theme.of(context).textTheme.bodyText1)),
    onTap: () {},
  ), // InkWell
), // Material
```

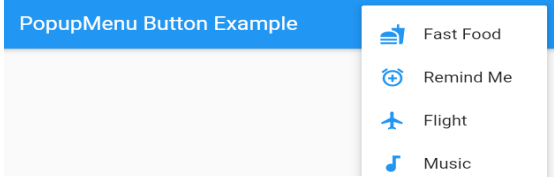
PopupMenuButton

- PopupMenuButton là một nút hiển thị menu khi được nhấn và sau đó khi gọi phương thức onSelected thì menu bị dừng hiển thị.
- Thường sử dụng với menu Settings để liệt kê tất cả các tùy chọn.
- Ví dụ: Các bước tạo PopupMenuButton
  - Tạo lớp TodoMenuItem chứa title và icon.
  - Tạo List<TodoMenuItem>.
  - Tạo PopupMenuButton.




```
class TodoMenuItem {
  final String title;
  final Icon icon;
  TodoMenuItem({this.title, this.icon});
}
```

PopupMenuButton



- Ví dụ: Các bước tạo PopupMenuButton
  - Tạo lớp TodoMenuItem chứa title và icon.
  - Tạo List<TodoMenuItem>.
  - Tạo PopupMenuButton.

```
class TodoMenuItem {
  final String title;
  final Icon icon;
  TodoMenuItem({this.title, this.icon});
}
```

```
List<TodoMenuItem> foodMenuList = [
  TodoMenuItem(title: 'Fast Food', icon: Icon(Icons.fastfood)),
  TodoMenuItem(title: 'Remind Me', icon: Icon(Icons.add_alarm)),
  TodoMenuItem(title: 'Flight', icon: Icon(Icons.flight)),
  TodoMenuItem(title: 'Music', icon: Icon(Icons.audiotrack)),
];
```

```
appBar: AppBar(
  title: const Text('PopupMenu Button Example'),
  actions: [
    PopupMenuButton<TodoMenuItem>(
      icon: Icon(Icons.view_list),
      onSelected: ((valueSelected) {
        print('valueSelected: ${valueSelected.title}');
      }),
      itemBuilder: (BuildContext context) {
        return foodMenuList.map((TodoMenuItem todoMenuItem) {
          return PopupMenuItem<TodoMenuItem>(
            value: todoMenuItem,
            child: Row(
              children: <Widget>[
                Icon(
                  todoMenuItem.icon.icon,
                  color: Colors.blue,
                ), // Icon
                Padding(
                  padding: EdgeInsets.all(8.0),
                ), // Padding
                Text(todoMenuItem.title),
              ], // <Widget>[]
            ), // Row
          ); // PopupMenuItem
        }).toList();
      },
    ), // PopupMenuButton
```

ButtonBar

- ButtonBar cung cấp sự linh hoạt để sắp xếp các nút trong một thanh hoặc một hàng.
- ButtonBar chứa ba thuộc tính: alignment, children và mainAxisSize.
  - alignment được sử dụng để trình bày tùy chọn căn chỉnh cho toàn bộ widget thanh nút.
  - children được sử dụng để lấy số lượng nút trong một thanh.
  - mainAxisSize được sử dụng để cung cấp không gian ngang cho thanh nút.

```
child: Container(
  color: Colors.white70,
  child: ButtonBar(
    alignment: MainAxisAlignment.spaceEvenly,
    children: <Widget>[
      IconButton(
        icon: Icon(Icons.map),
        onPressed: () {},
      ), // IconButton
      IconButton(
        icon: Icon(Icons.airport_shuttle),
        onPressed: () {},
      ), // IconButton
      IconButton(
        icon: Icon(Icons.brush),
        onPressed: () {},
      ), // IconButton
    ], // <Widget>[]
  ), // ButtonBar
), // Container
```