3. Kiểm thử ứng dụng app Flutter

[Nguyễn Thị Trang updated 7 months ago](https://docs.lumi.vn/doc/3-kiem-thu-ung-dung-app-flutter-mta0nwTZNp/history)  • [Comment](https://docs.lumi.vn/doc/3-kiem-thu-ung-dung-app-flutter-mta0nwTZNp) • [Viewed by 6 people](https://docs.lumi.vn/doc/3-kiem-thu-ung-dung-app-flutter-mta0nwTZNp/insights)

Kiểm tra(Testing) là một hoạt động được sử dụng để xác minh và xác thực một phần mềm hoặc ứng dụng không có lỗi và đáp ứng các yêu cầu của người dùng. Nó đảm bảo rằng kết quả thực tế phù hợp với kết quả mong đợi. Nó cũng giúp cải thiện phần mềm hoặc ứng dụng về hiệu quả, khả năng sử dụng và độ chính xác.

Kiểm tra là một trong những giai đoạn quan trọng nhất trong vòng đời phát triển ứng dụng để đảm bảo ứng dụng có chất lượng cao. Đây là giai đoạn tiêu tốn nhiều nhất trong quá trình phát triển ứng dụng hoặc phần mềm.

Framework Flutter cung cấp hỗ trợ tuyệt vời cho việc **kiểm tra tự động** một ứng dụng. Nói chung, kiểm thử tự động phân loại thành ba loại để kiểm tra hoàn toàn một ứng dụng. Chúng như sau:

1. Kiểm tra đơn vị(Unit Testing)
2. Kiểm tra widget(Widget Testing)
3. Tích hợp Thử nghiệm(Integration testing)

Sự cân bằng giữa các loại thử nghiệm khác nhau được đưa ra dưới đây:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Kiểm tra đơn vị** | **Kiểm tra widget** | **Tích hợp Thử nghiệm** |
| **Sự tự tin** | Thấp | Cao hơn | Cao nhất |
| **Bảo trì** | Tốn phí | Thấp cao hơn | Cao nhất |
| **Sự phụ thuộc ở các dữ án** | Vài cái | Hơn | Phần lớn |
| **Tốc độ thực thi** | Nhanh chóng | Nhanh chóng | Chậm |

Chúng tôi biết rằng, trong Flutter, mọi thứ đều là một widget. Vì vậy, ở đây, chúng ta sẽ thảo luận chi tiết về việc kiểm tra widget.

#1. Kiểm tra đơn vị

Đây là Phương thức dễ nhất để kiểm tra một ứng dụng hoặc phần mềm. Nó kiểm tra một hàm, phương thức hoặc lớp. Mục tiêu của kiểm thử đơn vị là đảm bảo tính đúng đắn của mã trong nhiều điều kiện khác nhau. Nói chung, kiểm thử đơn vị không tương tác với đầu vào của người dùng, hiển thị trên màn hình, đọc hoặc ghi dữ liệu từ đĩa và không sử dụng các phần phụ thuộc bên ngoài theo mặc định. Khi bạn sử dụng các phần phụ thuộc bên ngoài, chúng sẽ bị chế nhạo với các gói như Mockito.

#2. Kiểm tra widget

Kiểm tra widget được sử dụng để kiểm tra một widget. Mục tiêu của thử nghiệm này là để đảm bảo rằng giao diện người dùng của widget trông và tương tác với các widget khác như mong đợi. Quá trình kiểm tra widget tương tự như kiểm thử đơn vị, nhưng nó toàn diện hơn kiểm thử đơn vị. Thử nghiệm này liên quan đến nhiều lớp và yêu cầu một môi trường thử nghiệm để tìm ra nhiều lỗi hơn. Một widget, đang được thử nghiệm, có thể nhận và phản hồi các hành động và sự kiện của người dùng và có thể khởi tạo các widget con.

#3. Tích hợp Thử nghiệm

Kiểm thử tích hợp bao gồm cả kiểm tra ở trên cùng với các thành phần bên ngoài của ứng dụng. Nó xác thực một ứng dụng hoàn chỉnh hoặc một phần lớn của ứng dụng. Mục đích của kiểm tra tích hợp là để đảm bảo rằng tất cả các widget và dịch vụ hoạt động cùng nhau như mong đợi. Nó cũng có thể sử dụng để xác minh hiệu suất của ứng dụng. Nói chung, thử nghiệm tích hợp chạy trên các thiết bị thực như trình giả lập Android hoặc trình mô phỏng iOS.

Unit Testing

[Nguyễn Thị Trang updated 7 months ago](https://docs.lumi.vn/doc/1-unit-testing-IF0Kty7t87/history)  • [Comment](https://docs.lumi.vn/doc/1-unit-testing-IF0Kty7t87) • [Viewed by 5 people](https://docs.lumi.vn/doc/1-unit-testing-IF0Kty7t87/insights)

Làm thế nào để đảm bảo rằng ứng dụng của bạn tiếp tục hoạt động khi thêm nhiều tính năng hoặc thay đổi các tính năng cũ? Giải pháp đó chính là viết các test.

Unit tests là phương thức xác định những hành vi chức năng đơn lẻ, method hoặc class đơn lẻ. Một [test](https://pub.dev/packages/test) package cung cấp một core framework để thực thi viết các unit tests, và trong [flutter\_test](https://api.flutter.dev/flutter/flutter_test/flutter_test-library.html) package cung cấp thêm nhiều các tiện ích bổ sung cho kiểm thử các widget.

Công thức cho các tính năng lõi này được cung cấp bởi test package, theo các bước sau đây:

1. Thêm test hoặc flutter\_test dependency.
2. Tạo một file test.
3. Tạo một class cho test.
4. Viết một test cho class.
5. Gộp nhiều tests trong một nhóm.
6. Run các tests.

Để biết thêm nhiều tài liệu về test package, xem thêm tại link: [test package documentation](https://pub.dev/packages/test).

## **#Bước 1. Thêm test dependency**

Test package cung cấp chức năng chính (lõi) trong việc viết tests trong Dart. Đây là cách tiếp cận tốt nhất để viết các packages cho các ứng dụng web, server, và Flutter.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

dev\_dependencies:

test: <latest\_version>

## **#Bước 2. Tạo một file test**

Trong ví dụ, tạo 2 files: **counter.dart** và **counter\_test.dart**.

File **counter.dart** chức một class mà chúng ta muốn test và lưu trong **lib folder**. Tệp **counter\_test.dart** chứa các test của chính chúng và tồn tại bên trong **test folder.**

Thêm vào đó, các tệp test nên được lưu lại bên trong 1 **test folder**, vị trí của nó được lưu tại thư mục gốc của ứng dụngFlutter or package. Các file Test nên luôn luôn kết thúc **with \_test.dart**, điều này quy ước trong việc sử dụng test runner khi tìm kiếm các tests.

Khi chúng ta kết thúc, Cấu trúc thư mục nên được làm như sau:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

counter\_app/

lib/

counter.dart

test/

counter\_test.dart

## **# Bước 3. Tạo một class để test**

Bước tiếp theo, chúng ta cần một “unit” để test. Luôn nhớ rằng: “unit” là một tên khác cho một function, method, hoặc class. Ví dụ, tạo một Counter class trong file lib/counter.dart. Nó sẽ chịu trách nhiệm (tăng)incrementing và (giảm) decrementing (giảm) một giá trị bắt đầu bằng 0.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

class Counter {

int value = 0;

void increment() => value++;

void decrement() => value--;

}

**Chú ý:** Để đơn giản, bài hướng dẫn này sẽ không tiếp cận hướng “Test Driven Development”. Nếu bạn theo hướng driven thì bạn có thể theo hướng đó

## **#Bước 4. Viết một test cho class**

Trong file **counter\_test.dart**, viết một unit test đầu tiên. Tests được định nghĩa sử dụng top-level test function, và chúng ta có thể kiểm tra nếu kết quả chính xác hoặc nếu các kết quả chính xác bằng việc sử dụng top-level expect function. Cả hai phương pháp này đều trong **test package**.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

// Import the test package and Counter class

import 'package:counter\_app/counter.dart';

import 'package:test/test.dart';

void main() {

test('Counter value should be incremented', () {

final counter = Counter();

counter.increment();

expect(counter.value, 1);

});

}

## **#5. Gộp nhiều tests vào một group**

Nếu có nhiều tests tương tự nhau và liên quan đến nhau, thì gộp chúng lại vào trong một nhóm, tính năng này được cung cấp trong test package.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

import 'package:counter\_app/counter.dart';

import 'package:test/test.dart';

void main() {

group('Counter', () {

test('value should start at 0', () {

expect(Counter().value, 0);

});

test('value should be incremented', () {

final counter = Counter();

counter.increment();

expect(counter.value, 1);

});

test('value should be decremented', () {

final counter = Counter();

counter.decrement();

expect(counter.value, -1);

});

});

}

## **#6. Run the tests**

Bây giờ chúng ta có một class Counter class với các test được đặt trong đó, chúng ta có thể run tests.

### #Run tests sử dụng IntelliJ hoặc VSCode

Flutter plugins trong IntelliJ và VSCode hỗ trợ running các tests. Điều này là một lựa chon tốt nhât khi viết bởi vì nó cung cấp feedback loop nhanh nhất cũng như khả năng để set các điểm dừng (breakpoint).

* **IntelliJ**
  1. Mở file **counter\_test.dart**
  2. Chọn menu **Run**
  3. Click Run 'tests trong counter\_test.dart' option
  4. Ngoài ra có thể sử dụng các phím tắt tùy theo nền tảng của bạn.
* **VSCode**
  1. Mở file counter\_test.dart
  2. Chọn menu **Run**
  3. Click vào option Start Debugging
  4. Ngoài ra có thể sử dụng các phím tắt tùy theo nền tảng của bạn.

### #Run tests trong một terminal

Chúng ta có thể sử dụn một terminal để chạy các test bằng việc thực thi các câu lệnh sau từ thư mục gốc của dự án:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

flutter test test/counter\_test.dart

Để biết thêm nhiều options liên quan về unit tests, chúng ta có thể thực hiện câu lệnh sau:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

flutter test --help

Comment

2. Widget Testing

[Nguyễn Thị Trang updated 7 months ago](https://docs.lumi.vn/doc/2-widget-testing-dCLTRetWgI/history)  • [Comment](https://docs.lumi.vn/doc/2-widget-testing-dCLTRetWgI) • [Viewed by 3 people](https://docs.lumi.vn/doc/2-widget-testing-dCLTRetWgI/insights)

#Giới thiệu về Kiểm thử widget

Trong kiểm tra widget, bạn cần một số công cụ bổ sung được cung cấp bởi gói **flut\_test** . Gói này cung cấp các công cụ sau để kiểm tra widget.

* **WidgetTester:** Nó cho phép xây dựng và tương tác với các widget trong môi trường thử nghiệm.
* **testWidgets():** Phương thức này tự động tạo một WidgetTester cho mỗi trường hợp thử nghiệm. Nó được sử dụng như một hàm test () bình thường. Nó chấp nhận hai đối số: mô tả thử nghiệm và mã thử nghiệm.
* **Lớp Finder:** Nó được sử dụng để tìm kiếm các widget trong môi trường thử nghiệm.
* **Lớp Matcher:** Nó giúp xác minh xem một lớp Finder định vị một widget hay nhiều widget trong môi trường thử nghiệm.

Hãy để chúng ta tìm hiểu cách tất cả những điều trên phù hợp với nhau bằng các bước sau:

**Bước 1:** Thêm sự phụ thuộc của Flagship\_test.

Trong bước đầu tiên, chúng ta cần thêm một phần phụ thuộc flay\_test trong file **pubspec.yaml** . Theo mặc định, nó đã được thêm vào phần phụ thuộc.

**Bước 2:** Tạo một widget để kiểm tra.

Tiếp theo, chúng ta phải tạo một widget để thực hiện thử nghiệm. Đoạn mã dưới đây tạo một widget chứa tiêu đề và thông báo để hiển thị trên màn hình.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

class MyAppWidget extends StatelessWidget {

final String title;

final String message;

const MyWidget({

Key key,

@required this.title,

@required this.message,

}) : super(key: key);

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Flutter Testing Demo',

home: Scaffold(

appBar: AppBar(

title: Text(title),

),

body: Center(

child: Text(message),

),

),

);

}

}

**Bước 3:** Tạo thử nghiệm testWidgets.

Để kiểm tra widget, hãy sử dụng phương thức **testWidget()** . Phương thức này cho phép chúng ta xác định một thử nghiệm và chấp nhận hai đối số: mô tả thử nghiệm và mã thử nghiệm. Nó cũng tạo ra một **WidgetTester** để làm việc với widget. Đoạn mã sau xác minh rằng MyAppWidget hiển thị tiêu đề và tin nhắn.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

void main() {

testWidgets(' The widget contains a title and message', (WidgetTester tester) async {

// Write test code here.

});

}

**Bước 4:** Xây dựng widget bằng WidgetTester.

WidgetTester cung cấp phương thức **pumpWidget ()** để xây dựng và hiển thị widget được cung cấp. Nó tạo ra bản sao của MyAppWidget hiển thị ‘Ti’ và ‘Msg’ làm tiêu đề và thông điệp tương ứng. Đoạn mã sau đây giải thích rõ ràng hơn.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

void main() {

testWidgets(The widget contains a title and message', (WidgetTester tester) async {

await tester.pumpWidget(MyWidget(title: 'Ti', message: 'Msg'));

});

}

**Bước 5:** Tìm kiếm widget bằng Finder.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

Trong bước này, chúng ta sẽ sử dụng lớp Finder để tìm kiếm tiêu đề và tin nhắn trên cây widget. Nó cho phép chúng tôi xác minh rằng widget được hiển thị chính xác. Để làm điều này, chúng ta cần sử dụng phương thức **find.text ()** .

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

void main() {

testWidgets(The widget contains a title and message', (WidgetTester tester) async {

await tester.pumpWidget(MyWidget(title: 'Ti', message: 'Msg'));

final titleFinder = find.text('Ti');

final messageFinder = find.text('Msg');

});

}

**Bước 6:** Xác minh widget bằng Matcher.

Cuối cùng, chúng ta cần xác minh tin nhắn văn bản xuất hiện trên màn hình bằng cách sử dụng lớp Matcher. Nó đảm bảo rằng widget xuất hiện trên màn hình chính xác một lần. Chúng ta có thể xem đoạn mã sau để hiểu nó.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

void main() {

testWidgets(The widget contains a title and message', (WidgetTester tester) async {

await tester.pumpWidget(MyWidget(title: 'Ti', message: 'Msg'));

final titleFinder = find.text('Ti');

final messageFinder = find.text('Msg');

expect(titleFinder, findsOneWidget);

expect(messageFinder, findsOneWidget);

});

}

Bây giờ, chúng ta sẽ xem ví dụ làm việc để hiểu khái niệm kiểm tra widget. Đầu tiên, tạo một dự án trong Android Studio và điều hướng đến thư mục thử nghiệm của thư mục dự án. Bây giờ, hãy mở tệp **widget\_test.dart** và thay thế mã sau:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

import 'package:flutter/material.dart';

import 'package:flutter\_test/flutter\_test.dart';

void main() {

testWidgets(The widget contains a title and message', (WidgetTester tester) async {

// Create the widget by telling the tester to build it.

await tester.pumpWidget(MyWidget(title: 'Ti', message: 'Msg'));

// Create the Finders.

final titleFinder = find.text('Ti');

final messageFinder = find.text('Msg');

expect(titleFinder, findsOneWidget);

expect(messageFinder, findsOneWidget);

});

}

class MyAppWidget extends StatelessWidget {

final String title;

final String message;

const MyAppWidget({

Key key,

@required this.title,

@required this.message,

}) : super(key: key);

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Flutter Testing Demo',

home: Scaffold(

appBar: AppBar(

title: Text(title),

),

body: Center(

child: Text(message),

),

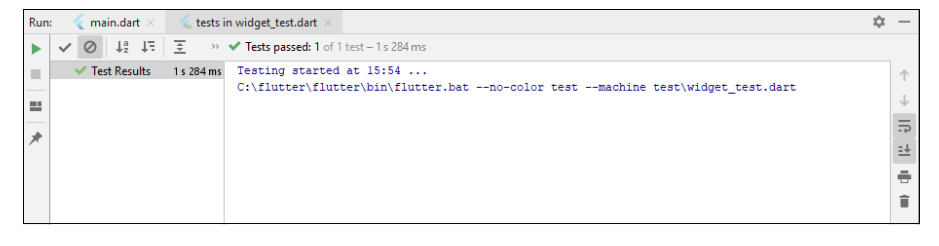
),

);

}

}

Để thực hiện kiểm tra, hãy chuyển đến menu **Chạy** và chọn tùy chọn “kiểm tra trong widget\_test.dart”. Nó sẽ chạy thử nghiệm và cho kết quả như màn hình sau:



Integration testing

[Nguyễn Thị Trang updated 7 months ago](https://docs.lumi.vn/doc/3-integration-testing-xUxPgxG0um/history)  • [Comment](https://docs.lumi.vn/doc/3-integration-testing-xUxPgxG0um) • [Viewed by 3 people](https://docs.lumi.vn/doc/3-integration-testing-xUxPgxG0um/insights)

Kiểm thử đơn vị và kiểm thử widget là kiểm thử các class, chức năng hoặc các widget độc lập. Thế nhưng làm thế nào để các class, các function đó làm việc cùng nhau hoặc ghi lại hiệu suất của ứng dụng đang chạy trên thiết bị thực. Và để làm được nhiệm vụ này thì phải thực hiện bởi kiểm thử tích hợp ***integration tests*.**

Kiểm thử tích hợp được viết dựa trên gói [integration\_test](https://github.com/flutter/flutter/tree/main/packages/integration_test) package, cung cấp bởi SDK.

Ở phương thức kiểm thử này, chúng ta học cách để kiểm thức ứng dụng counter. Nó chứng minh được rằng làm thế nào để cài đặt kiểm thử tích hợp, làm thế nào để xác thực các text đặc biệt hiển thị trên app và làm thế nào để tap vào các widget đặc biệt, và làm thế nào để chạy kiểm thử tích hợp.

Các bước thực hiện kiểm thử tích hợp như sau:

1. Tạo một ứng dụng test
2. Thêm một test dependency

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

integration\_test

1. Tạo các file test
2. Viết kiểm thử tích hợp.
3. Và chạy kiểm thử tích hợp.

#Bước 1. Create an app to test

Đầu tiên, tạo một ứng dụng để thực hiện kiểm thử. Trong ví dụ dưới đây, kiểm thử ứng dụng **counter** được thực hiện bởi các câu lệnh flutter. Ứng dụng cho phép 1 user tap vào một button để tăng số đếm.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {

const MyApp({super.key});

@override

Widget build(BuildContext context) {

return const MaterialApp(

title: 'Counter App',

home: MyHomePage(title: 'Counter App Home Page'),

);

}

}

class MyHomePage extends StatefulWidget {

const MyHomePage({super.key, required this.title});

final String title;

@override

State<MyHomePage> 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>[

const Text(

'You have pushed the button this many times:',

),

Text(

'$\_counter',

// Provide a Key to this specific Text widget. This allows

// identifying the widget from inside the test suite,

// and reading the text.

key: const Key('counter'),

style: Theme.of(context).textTheme.headline4,

),

],

),

),

floatingActionButton: FloatingActionButton(

// Provide a Key to this button. This allows finding this

// specific button inside the test suite, and tapping it.

key: const Key('increment'),

onPressed: \_incrementCounter,

tooltip: 'Increment',

child: const Icon(Icons.add),

),

);

}

}

#Bước 2. Thêm integration\_test dependency

Bước tiếp theo, sử dụng integration\_test và các gói flutter\_test packages để viết các kiểm thử tích hợp. Thêm các dependencies vào mục **dev\_dependencies section** của file **pubspec.yaml**, xác định vị trí Flutter SDK trpopackage.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

dev\_dependencies:

integration\_test:

sdk: flutter

flutter\_test:

sdk: flutter

#Bước 3. Tạo các file tests

Tạo một mục mới, **integration\_test**, với một tệp rỗng tên là: **app\_test.dart**:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

counter\_app/

lib/

main.dart

integration\_test/

app\_test.dart

#Bước 4. Viết kiểm thử tích hợp

Bây giờ chúng ta có thể viết các test. Việc này liên quan đến 3 bước sau:

1. Khởi tạo một service đơn lẻ nhằm thực thi các test trong môi trường vật lý

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

IntegrationTestWidgetsFlutterBinding

1. Tương tác và kiểm thử các widget sử dụng class

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

WidgetTester

1. Test các kịch bản quan trọng.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

import 'package:flutter\_test/flutter\_test.dart';

import 'package:integration\_test/integration\_test.dart';

import 'package:counter\_app/main.dart' as app;

void main() {

IntegrationTestWidgetsFlutterBinding.ensureInitialized();

group('end-to-end test', () {

testWidgets('tap on the floating action button, verify counter',

(tester) async {

app.main();

await tester.pumpAndSettle();

// Verify the counter starts at 0.

expect(find.text('0'), findsOneWidget);

// Finds the floating action button to tap on.

final Finder fab = find.byTooltip('Increment');

// Emulate a tap on the floating action button.

await tester.tap(fab);

// Trigger a frame.

await tester.pumpAndSettle();

// Verify the counter increments by 1.

expect(find.text('1'), findsOneWidget);

});

});

}

#  
Bước 5. Run kiểm thử tích hợp

Quy trình run các kiểm thử tích hợp phụ thuộc vào nên tảng mà chúng ta đang kiểm thư. Chúng ta có thể kiểm thử trên 2 nền tảng mobile hoặc nền tảng web.

#5a. Mobile

Để thực hiện kiểm thử trên thiết bị iOS / Android thực, đầu tiên cần kết nối các thiết bị và run theo các câu lệnh sau đây ở root của dự án:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

flutter test integration\_test/app\_test.dart

Hoặc, chúng ta có thể run trực tiếp tất cả các kiểm thử tích hợp:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

flutter test integration\_test

Câu lệnh runs ứng dụng và kiểm thử tích hợp trên thiết bị mục tiêu. Để biết nhiều thông tin hơn, có thể vào trang [Integration testing](https://docs.flutter.dev/testing/integration-tests) để tìm hiểu.

#5b. Web

Để bắt đầu thực hiện kiểm thử tích hợp trên ứng dụng web browser, thì phải download: [Download ChromeDriver](https://chromedriver.chromium.org/downloads).

Tiếp theo, tạo một thư mục mới có tên **test\_driver** chứa một file mới có tên là **integration\_test.dart**:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

import 'package:integration\_test/integration\_test\_driver.dart';

Future<void> main() => integrationDriver();

Khởi tạo chromedriver với câu lệnh sau:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

chromedriver --port=4444

Từ thư mục gốc của dự án, chạy các câu lệnh sau:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

flutter drive \

--driver=test\_driver/integration\_test.dart \

--target=integration\_test/app\_test.dart \

-d chrome

Đối với kiển thử ứng không có giao diện đồ họa người dùng (headless), chúng ta cũng có thể run flutter drive với **web-server** cũng như với thiết bị mục tiêu được xác định như sau:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

flutter drive \

--driver=test\_driver/integration\_test.dart \

--target=integration\_test/app\_test.dart \

-d web-server

Các hàm và method của framework Testing Flutter Apps

[Đỗ Thị Tân updated 6 months ago](https://docs.lumi.vn/doc/4-cac-ham-va-method-cua-framework-testing-flutter-apps-2qcDZh4dP7/history)  • [Comment](https://docs.lumi.vn/doc/4-cac-ham-va-method-cua-framework-testing-flutter-apps-2qcDZh4dP7) • [Viewed by 3 people](https://docs.lumi.vn/doc/4-cac-ham-va-method-cua-framework-testing-flutter-apps-2qcDZh4dP7/insights)

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

import 'package:flutter/material.dart';

import 'package:flutter\_test/flutter\_test.dart';

import 'package:integration\_test/integration\_test.dart';

import 'package:lumi\_life/main.dart' as app;

void main() {

IntegrationTestWidgetsFlutterBinding.ensureInitialized();

testWidgets("login screen test", (WidgetTester tester) async {

app.main();

await tester.pumpAndSettle();

final emailField = find.byType(TextFormField).first;

final passwordField = find.byType(TextFormField).at(1);

await tester.enterText(emailField, 'nghinv@lumi.biz');

await tester.enterText(passwordField, '123456');

await tester.tap(find.byType(ElevatedButton));

await Future.delayed(const Duration(seconds: 20), () {});

await tester.pumpAndSettle();

expect(find.byType(Text).first, findsOneWidget);

});

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

IntegrationTestWidgetsFlutterBinding.ensureInitialized();

* **IntegrationTestWidgetsFlutterBinding** là một lớp được sử dụng để kết nối giữa kiểm thử tích hợp và ứng dụng. Phương thức **ensureInitialized()** được gọi trên lớp này để đảm bảo rằng đối tượng **IntegrationTestWidgetsFlutterBinding** đã được khởi tạo trước khi thực hiện bất kỳ kiểm thử nào.
* Phương thức **ensureInitialized()** được gọi trong hàm **main()**
* Khi chạy integration test bằng Flutter, phương thức **ensureInitialized()** được gọi để đảm bảo rằng các **widget** và tài nguyên được tải và sẵn sàng trước khi bắt đầu thực hiện kiểm thử. Nếu không gọi phương thức này, kiểm thử sẽ không hoạt động đúng và có thể gây ra các lỗi hoặc sự cố.

Cấu trúc chung của một test sử dụng phương thức **testWidgets()** như sau:

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

testWidgets('test description', (WidgetTester tester) async {

// Khởi tạo các widget

// Thao tác với các widget

// Kiểm tra trạng thái của các widget

// Kiểm tra các tương tác giữa các widget

});

Trong đó, tham số đầu tiên của phương thức **testWidgets()** là mô tả của test, tham số thứ hai là một hàm **callback**, trong đó bạn có thể sử dụng tham số **WidgetTester** để thao tác với các widget, kiểm tra trạng thái của chúng và kiểm tra các tương tác giữa các widget.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

app.main();

dùng để khởi chạy ứng dụng.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

await tester.pumpAndSettle();

await tester.enterText(emailField, 'nghinv@lumi.biz');

Trong context của Integration Testing trong Flutter, **tester** là một đối tượng có kiểu **WidgetTester**, cung cấp các phương thức để tương tác với **widgets** và kiểm tra trạng thái của chúng. Các phương thức này bao gồm **pumpWidget**, **tap**, **enterText**, **pump**, **pumpAndSettle**, **waitFor**, và nhiều phương thức khác. Bằng cách sử dụng **tester**, ta có thể thực hiện các hoạt động trên **widgets** và kiểm tra xem chúng có hoạt động như mong đợi hay không.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

await Future.delayed(const Duration(seconds: 20), () {});

Sử dụng phương thức **delayed** của lớp **Future** để tạm dừng thực thi trong **20 giây** trước khi tiếp tục thực hiện các hành động khác.

Cho phép chương trình tạm dừng thực thi trong 20 giây (do đối số đầu tiên là **const Duration(seconds: 20)**) sau đó tiếp tục thực hiện các hành động khác được định nghĩa trong hàm truyền vào là () {}. Hàm truyền vào này không làm gì cả

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

expect(find.byType(Text).first, findsOneWidget);

một câu lệnh kiểm tra dùng để đảm bảo rằng có đúng một widget được tìm thấy bởi **find.byType(Text)** trong **widget tree** của ứng dụng.

**find.byType(Text)** trả về một **Finder** object, được sử dụng để tìm các **widgets** trong **widget tree** của ứng dụng. **.first** được gọi để trả về **widget** đầu tiên được tìm thấy bởi Finder . **findsOneWidget**

là một matcher được sử dụng để kiểm tra xem **widget** được tìm thấy có đúng một **widget** duy nhất trong **widget tree** của ứng dụng hay không.

## **#Screenshots**

Bạn có thể sử dụng integration\_test để chụp ảnh màn hình giao diện người dùng được hiển thị trên thiết bị di động hoặc trình duyệt Web tại một thời điểm cụ thể trong quá trình kiểm tra.

Tính năng này hiện được hỗ trợ trên Android, iOS và Web.

**integration\_test/screenshot\_test.dart**

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZig

void main() {

final binding = IntegrationTestWidgetsFlutterBinding.ensureInitialized()

as IntegrationTestWidgetsFlutterBinding;

testWidgets('screenshot', (WidgetTester tester) async {

// Build the app.

app.main();

// This is required prior to taking the screenshot (Android only).

await binding.convertFlutterSurfaceToImage();

// Trigger a frame.

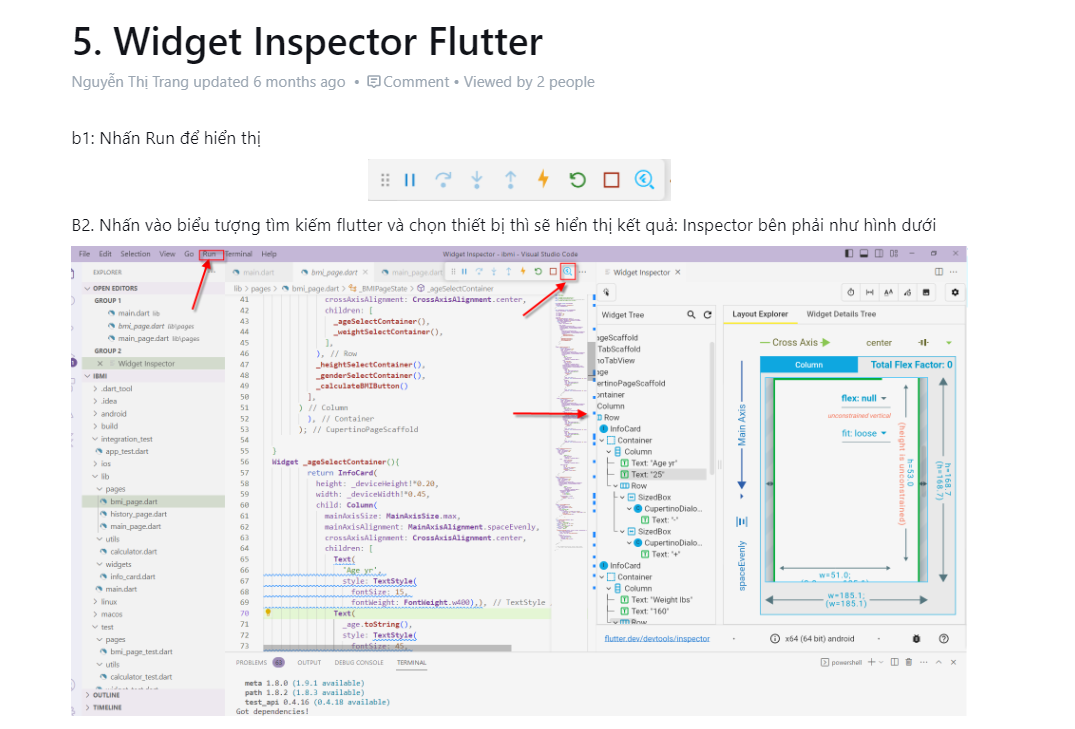
await tester.pumpAndSettle();

await binding.takeScreenshot('screenshot-1');

});

}

Comment



3/4 Demo Automation Tetsing

[Nguyễn Thị Trang updated 6 months ago](https://docs.lumi.vn/doc/1-34-demo-automation-tetsing-AxwJvKqaOU/history)  • [Comment](https://docs.lumi.vn/doc/1-34-demo-automation-tetsing-AxwJvKqaOU) • [Viewed by 3 people](https://docs.lumi.vn/doc/1-34-demo-automation-tetsing-AxwJvKqaOU/insights)

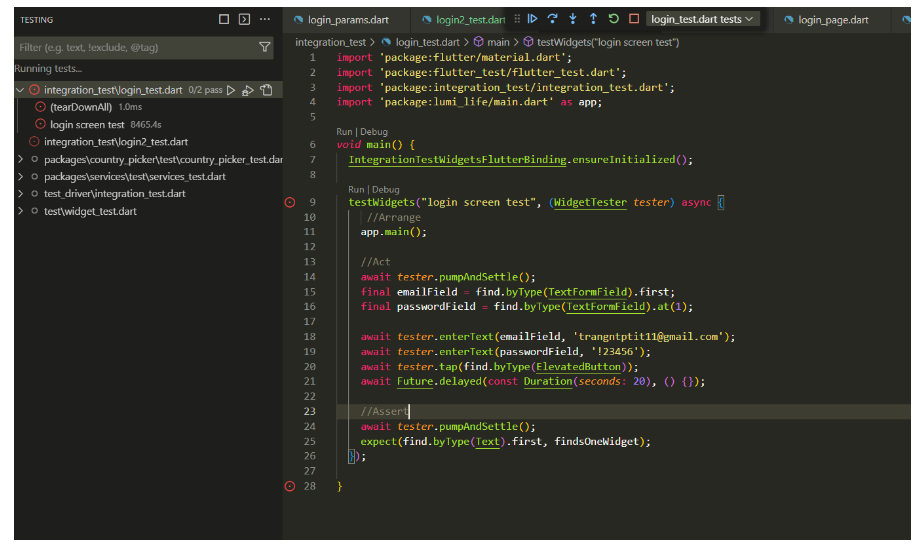
## **#1. Lọc testcase**

<https://test.management.lumi.vn/squash/campaign-workspace/campaign/374/dashboard>

tài khoản: trangnt/lumivn274

## **#2. Viết Scripts**

### #2.1 Viết scipts cho chức năng login



### #2.2 Viết scripts cho chức năng đăng ký

## #3. Một số vấn đề

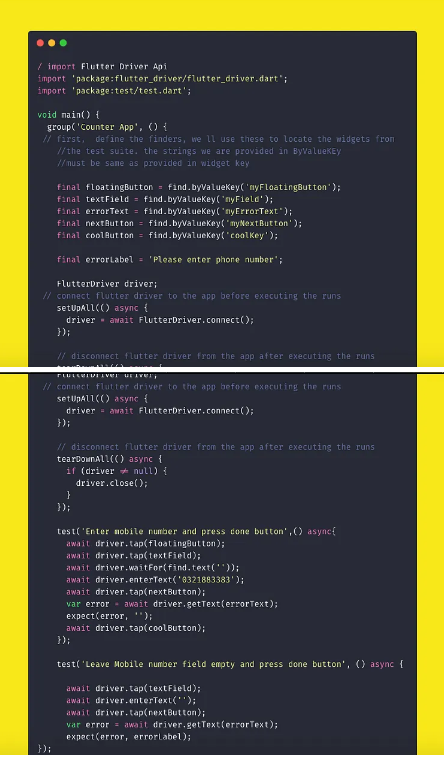
1. Tìm thấy một thư viện mới: **flutter driver**

<https://docs.flutter.dev/testing/integration-tests/migration>



 Thư viện mới flutter driver khá giống với cái mô hình

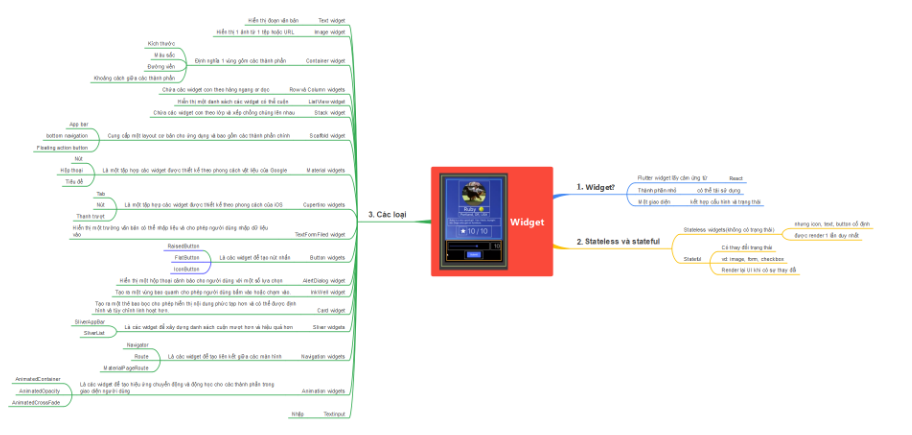
<https://medium.com/@syedanum21/flutter-flutter-driver-integration-testing-flutter-app-automation-ed71fb32622a>



Danh sách các loại widget trong Flutter

[Nguyễn Thị Trang updated 5 months ago](https://docs.lumi.vn/doc/2-danh-sach-cac-loai-widget-trong-flutter-Lm809a2Iyu/history)  • [Comment](https://docs.lumi.vn/doc/2-danh-sach-cac-loai-widget-trong-flutter-Lm809a2Iyu) • [Viewed by 3 people](https://docs.lumi.vn/doc/2-danh-sach-cac-loai-widget-trong-flutter-Lm809a2Iyu/insights)

|  |  |
| --- | --- |
| **STT** | **Loại widget** |
| 1 | Text |
| 2 | Column, Row |
| 3 | Stack |
| 4 | Container |
| 5 | SizedBox |
| 6 | SingleChildScrollView |
| 7 | Expanded |
| 8 | Image |
| 9 | IconButton |
| 10 | FlatButton |
| 11 | SelectableText |
| 12 | DataTable |
| 13 | Slider |
| 14 | AlertDialog |
| 15 | DraggableScrollableSheet |
| 16 | ColorFiltered |
| 17 | ToggleButtons |
| 18 | DefaultTabController & TabBar |



**Video giới thiệu về những widget**

<https://www.youtube.com/watch?v=TtaRSaUvji0&list=PLv6GftO355AsxyLjGVkpOmN8DUbcPdIBv&index=11>

# #I. Material Widget

– Là widget rất liện lợi, cung cấp các widget cho việc xây dựng ứng dụng sử dụng thư viện Material Design UI của google.  
– Widget này được sử dụng trong hàm build đầu tiên của hầu hết các ứng dụng.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

@override

Widget build(BuildContext context) {

return MaterialApp(

// Tạo Title của AppBar

title: 'Flutter Demo',

// false : tắt label "Debug" bên phải, default: true

debugShowCheckedModeBanner: false,

// Xây dựng Theme

theme: ThemeData(

primarySwatch: Colors.blue,

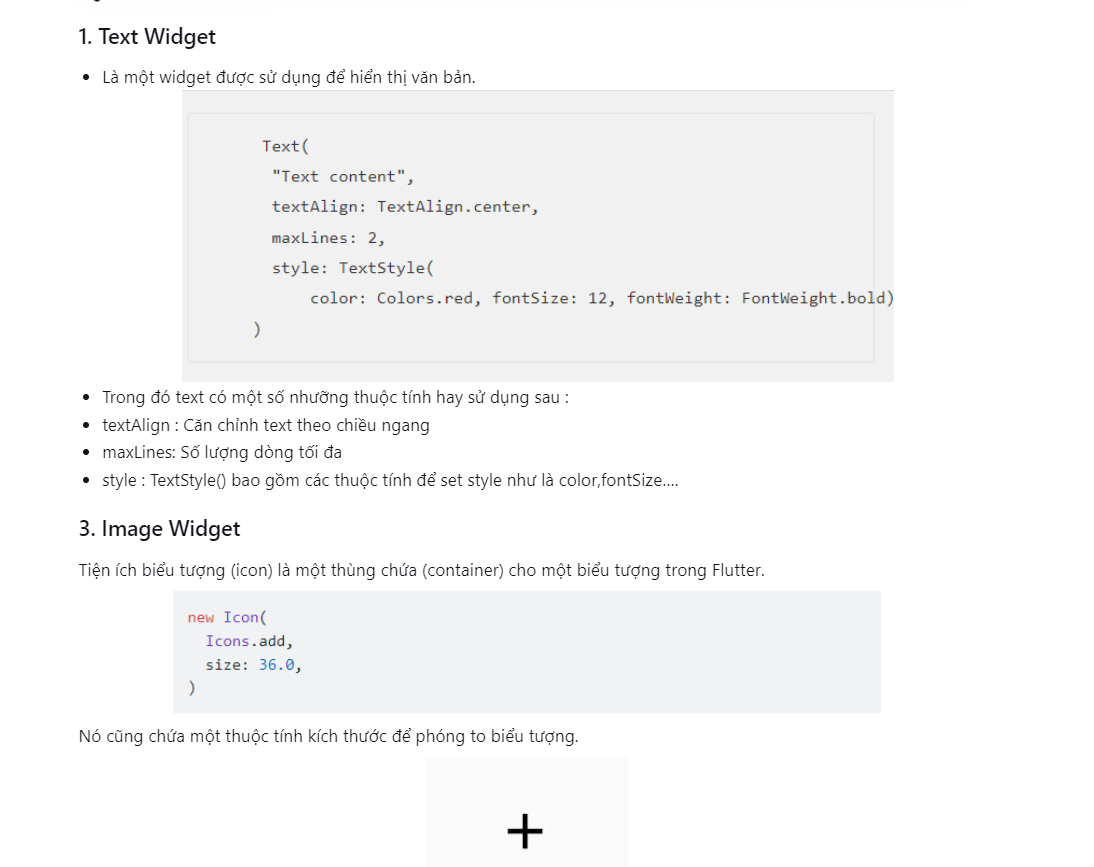
visualDensity: VisualDensity.adaptivePlatformDensity,

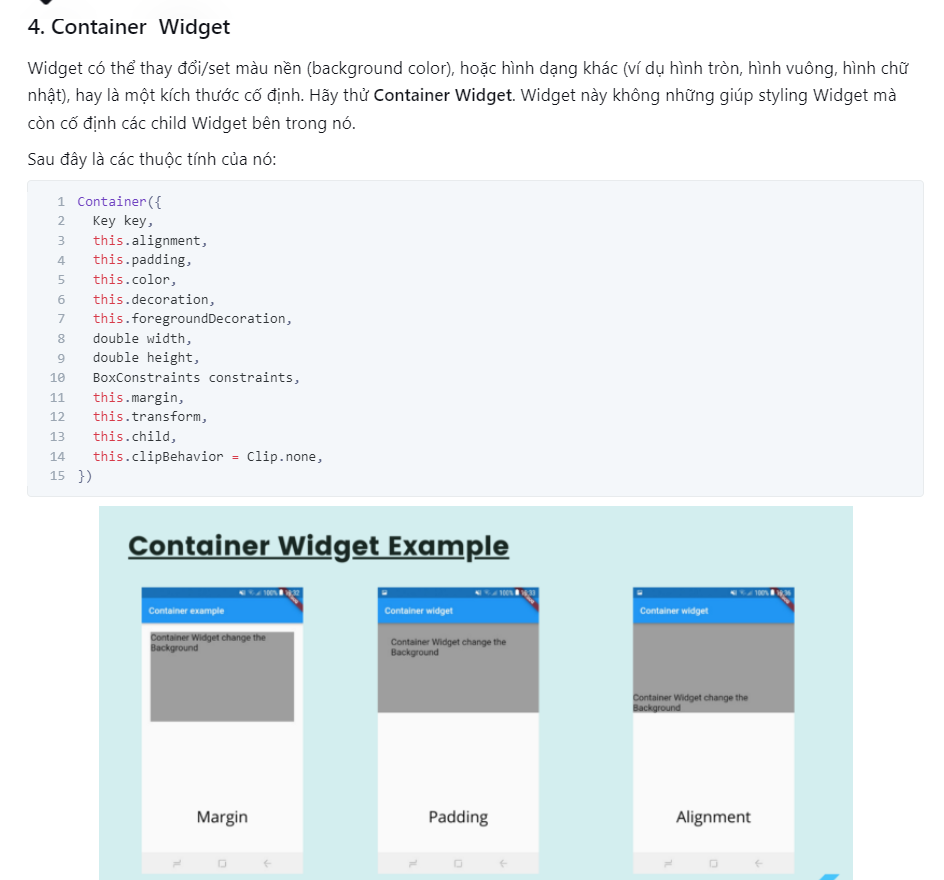
),

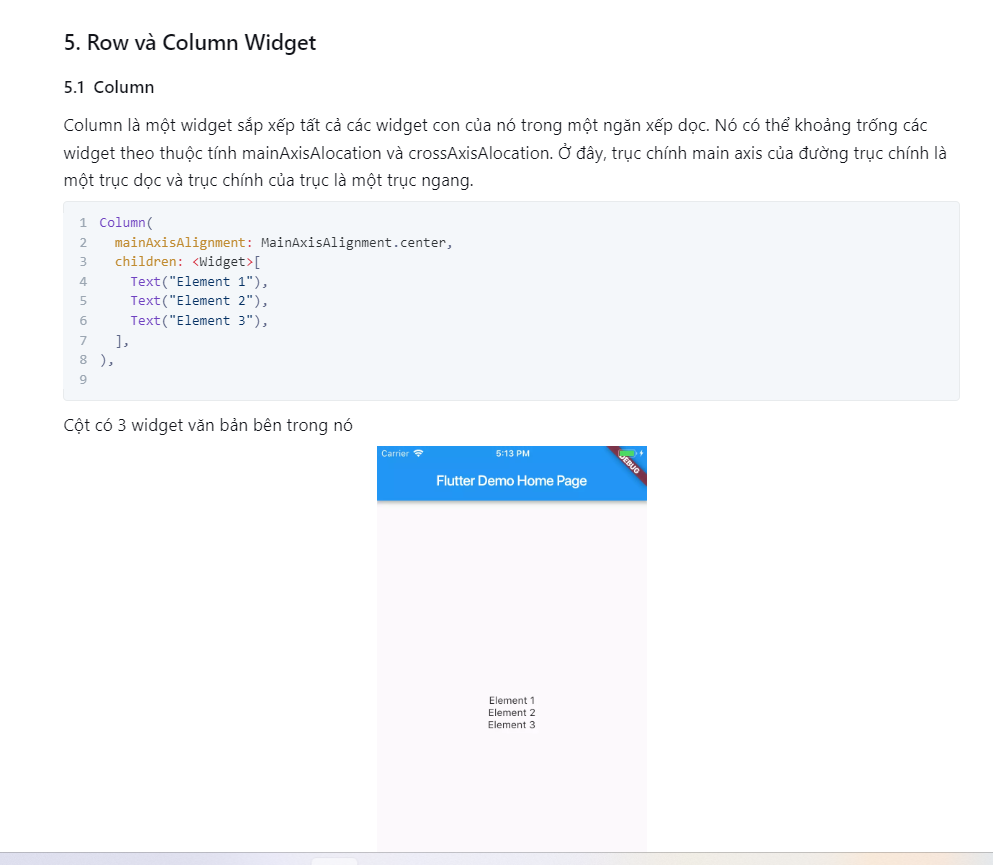
// Liên kết với Widget con qua từ khoá home:

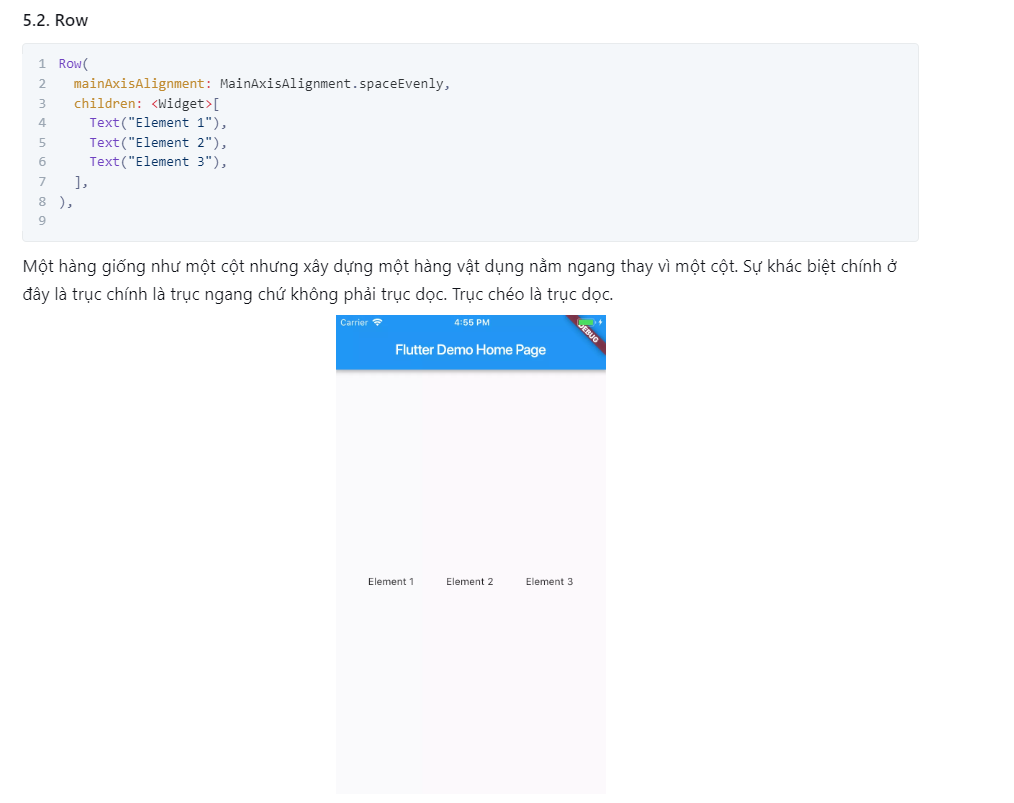
home: MyHomePage(),

);









## #. **ListView, GridView, ScrollView** Widget

– Đây là những widget dùng để chứa các widget con mà có thể scroll theo màn hình theo chiều dọc và chiều ngang.  
– Liên kết với các widget con qua thuộc tính : children

Bạn có thể xem chi tiết cách sử dụng từng loại : [ListView](https://api.flutter.dev/flutter/widgets/ListView-class.html), [GridView](https://api.flutter.dev/flutter/widgets/GridView-class.html), [ScrollView](https://api.flutter.dev/flutter/widgets/ScrollView-class.html)

## #7. Stack Widget

Một Stack được sử dụng để chồng lấp các widget, giống như một nút trên gradient nền. Stack là một widget rất quan trọng và chúng ta sẽ đi sâu vào chi tiết hơn trong một bài viết trong tương lai.

Đây là những widgets cơ bản cần có để tạo ra một ứng dụng hoạt động tốt trong Flutter. Flutter cũng cho phép bạn tạo các widget của riêng mình nếu bạn cần thêm bất kỳ chức năng nào khác hoặc nếu bạn muốn sử dụng lại một số mẫu widget lặp lại.

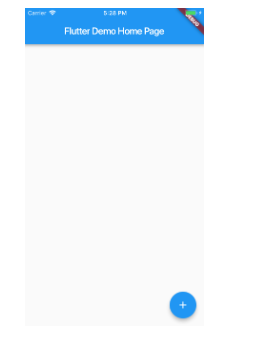
## #8. Scaffold Widget

Là Widget cung cấp cấu trúc layout cho màn hình ứng dụng.

Widget Scaffold là một khung để thêm các yếu tố thiết kế widget phổ biến như: Appbars, Drawers, Floating Action Buttons, Bottom Navigation, …

|  |  |
| --- | --- |
| **1**  **2**  **3**  **4**  **5**  **6**  **7**  **8**  **9**  **10**  **11**  **12** | **@override**  **Widget build(BuildContext context) {**  **return Scaffold(**  **// Tạo Appbar**  **appBar: AppBar(**  **title: Text(widget.title),**  **),**  **// Liên kết với Widget con qua thuộc tính body:**  **body: Center(**  **//......**  **),**  **}** |

Ngoài ra còn có các thuộc tính trong Scaffold . [Xem thêm](https://api.flutter.dev/flutter/material/Scaffold-class.html)



## #TextFormFiled Widget

Mặc dù chúng ta đã có Form, nhưng chưa có cách nào để người dùng nhập dữ liệu. Và ta cần tới TextFormField. TextFormField widget tạo ra một text field dạng material design và có thể hiển thị lỗi khi dữ liệu sai.

Xác thực đầu vào bằng cách cung cấp một hàm validator() cho TextFormField. Nếu thông tin đầu vào của người dùng không hợp lệ, hàm validator() sẽ trả về một chuỗi chứa thông báo lỗi. Nếu không có lỗi, validator phải trả về null.

Đối với ví dụ này, hãy tạo validator để đảm bảo TextFormField không trống. Nếu nó trống, hãy trả về một thông báo lỗi.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

TextFormField(

// The validator receives the text that the user has entered.

validator: (value) {

if (value == null || value.isEmpty) {

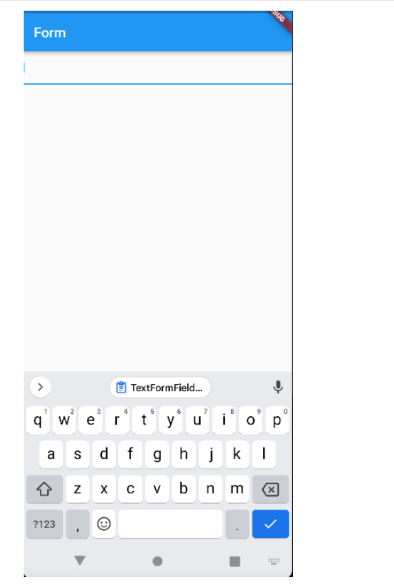
return 'Please enter some text';

}

return null;

},

),



## #10. Widget hộp thoại cảnh báo

### #10.1 AlertDialog Widget

Cảnh báo này thông báo cho người dùng về thông tin mới, chẳng hạn như thay đổi trong ứng dụng, về các tính năng mới, tình huống khẩn cấp cần xác nhận hoặc như một thông báo xác nhận cho người dùng rằng hành động có thành công hay không. Ví dụ sau giải thích việc sử dụng các cảnh báo cơ bản.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

showAlertDialog(BuildContext context) {

// Create button

Widget okButton = FlatButton(

child: Text("OK"),

onPressed: () {

Navigator.of(context).pop();

},

);

// Create AlertDialog

AlertDialog alert = AlertDialog(

title: Text("Simple Alert"),

content: Text("This is an alert message."),

actions: [

okButton,

],

);

// show the dialog

showDialog(

context: context,

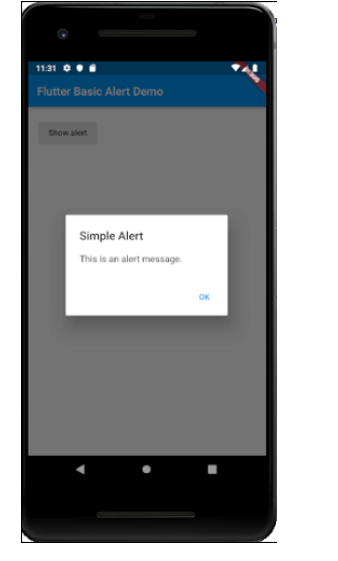
builder: (BuildContext context) {

return alert;

},

);

}



**#10.2. TextField AlertDialog**

AlertDialog này làm cho nó có thể chấp nhận đầu vào của người dùng. Trong ví dụ sau, chúng ta sẽ thêm đầu vào trường văn bản trong hộp thoại cảnh báo. Mở tệp main.dart và chèn code sau.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

class TextFieldAlertDialog extends StatelessWidget {

TextEditingController \_textFieldController = TextEditingController();

\_displayDialog(BuildContext context) async {

return showDialog(

context: context,

builder: (context) {

return AlertDialog(

title: Text('TextField AlertDemo'),

content: TextField(

controller: \_textFieldController,

decoration: InputDecoration(hintText: "TextField in Dialog"),

),

actions: <Widget>[

new FlatButton(

child: new Text('SUBMIT'),

onPressed: () {

Navigator.of(context).pop();

},

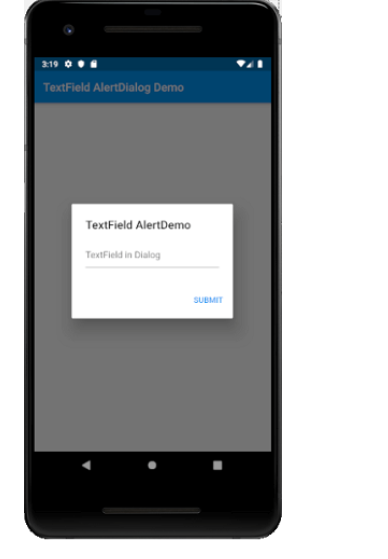
)

],

);

});

}



**#10.3 Confirmation AlertDialog**

Hộp thoại cảnh báo xác nhận thông báo cho người dùng xác nhận một lựa chọn cụ thể trước khi chuyển tiếp trong ứng dụng. Ví dụ: khi người dùng muốn xóa hoặc xóa một số liên lạc khỏi sổ địa chỉ.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

enum ConfirmAction { Cancel, Accept}

Future<ConfirmAction> \_asyncConfirmDialog(BuildContext context) async {

return showDialog<ConfirmAction>(

context: context,

barrierDismissible: false, // user must tap button for close dialog!

builder: (BuildContext context) {

return AlertDialog(

title: Text('Delete This Contact?'),

content: const Text(

'This will delete the contact from your device.'),

actions: <Widget>[

FlatButton(

child: const Text('Cancel'),

onPressed: () {

Navigator.of(context).pop(ConfirmAction.Cancel);

},

),

FlatButton(

child: const Text('Delete'),

onPressed: () {

Navigator.of(context).pop(ConfirmAction.Accept);

},

)

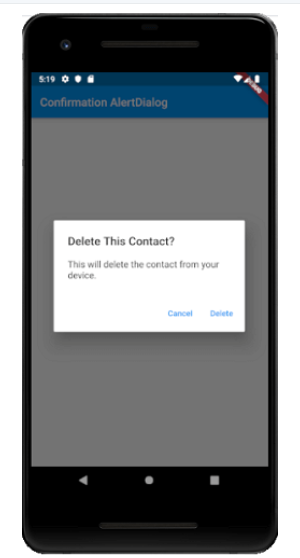
],

);

},

);

}



**Chọn option AlertDialog**

Loại hộp thoại cảnh báo này hiển thị danh sách các mục, sẽ có hành động ngay lập tức khi được chọn.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

enum Product { Apple, Samsung, Oppo, Redmi }

Future<Product> \_asyncSimpleDialog(BuildContext context) async {

return await showDialog<Product>(

context: context,

barrierDismissible: true,

builder: (BuildContext context) {

return SimpleDialog(

title: const Text('Select Product '),

children: <Widget>[

SimpleDialogOption(

onPressed: () {

Navigator.pop(context, Product.Apple);

},

child: const Text('Apple'),

),

SimpleDialogOption(

onPressed: () {

Navigator.pop(context, Product.Samsung);

},

child: const Text('Samsung'),

),

SimpleDialogOption(

onPressed: () {

Navigator.pop(context, Product.Oppo);

},

child: const Text('Oppo'),

),

SimpleDialogOption(

onPressed: () {

Navigator.pop(context, Product.Redmi);

},

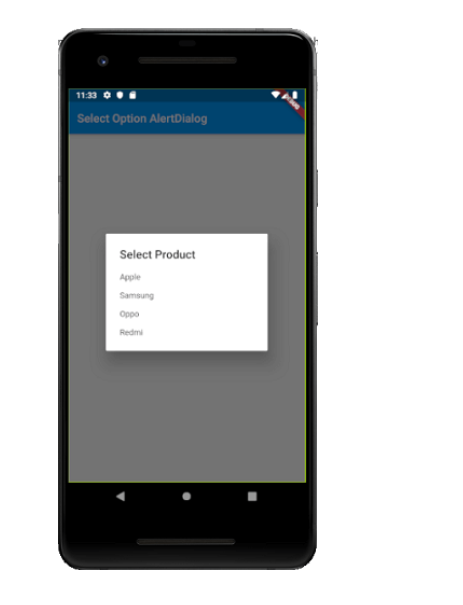
child: const Text('Redmi'),

),

],

);

});



## #12. Button Widget

Trong Flutter có sẵn rất nhiều Widget là các button. Người lập trình cũng có thể tự tạo những button riêng.  
– Bạn có thể đặt tên cho button qua – child: Text(“Tên Button”),  
– Hàm thực thi sau khi nhấn nút – onPressed: (){ // Hàm thực thi }

### #12.1. Flat Button (Nút phẳng)

Nó là một **nút nhãn văn bản** không có nhiều trang trí và hiển thị **mà không có bất kỳ độ cao nào** . Nút phẳng có hai thuộc tính bắt buộc là: **child và onPressed()** .

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

child: FlatButton(

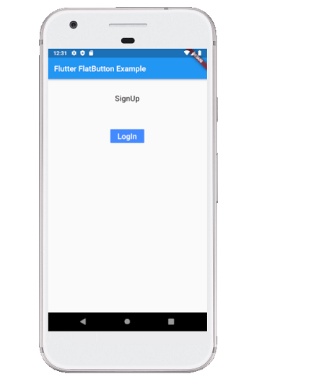
child: Text('LogIn', style: TextStyle(fontSize: 20.0),),

color: Colors.blueAccent,

textColor: Colors.white,

onPressed: () {},

),



**#12.2. Raised Button (Nút nâng)**

Nó là một nút, dựa trên vật liệu vật liệu và có **thân hình chữ nhật** . Nó tương tự như một nút phẳng, nhưng nó **có độ cao(elevation)** sẽ tăng lên khi nút được nhấn. Nó thêm thứ nguyên cho giao diện người dùng dọc theo trục Z. Nó có một số thuộc tính như màu văn bản, hình dạng, phần đệm, màu nút, màu của nút khi bị tắt, thời gian animation, độ cao, v.v.

Nút này có **hai chức năng gọi lại**.

**onPressed ():** Nó được kích hoạt khi nhấn nút.

**onLongPress ():** Nó được kích hoạt khi nhấn và giữ nút.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

RaisedButton(

child: Text("Click Here", style: TextStyle(fontSize: 20),),

onPressed: \_changeText,

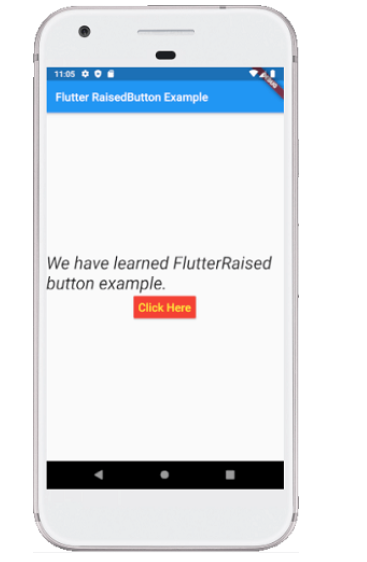
color: Colors.red,

textColor: Colors.yellow,

padding: EdgeInsets.all(8.0),

splashColor: Colors.grey,

)



**#12.3. Floating Button (Nút nổi)**

Nút FAB là một **nút biểu tượng hình tròn** kích hoạt hành động chính trong ứng dụng của chúng ta. Nó là nút được sử dụng nhiều nhất trong các ứng dụng hiện nay.

Có hai loại Nút hành động nổi:

**FloatingActionButton:** Nó tạo một nút nổi hình tròn đơn giản với một widget con bên trong nó. Nó phải có một tham số chile để hiển thị một widget.

**FloatingActionButton.extended:** Nó tạo ra một nút nổi rộng cùng với một biểu tượng và nhãn bên trong nó. Thay vì một chile, nó sử dụng các nhãn và các thông số biểu tượng.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

floatingActionButton: FloatingActionButton(

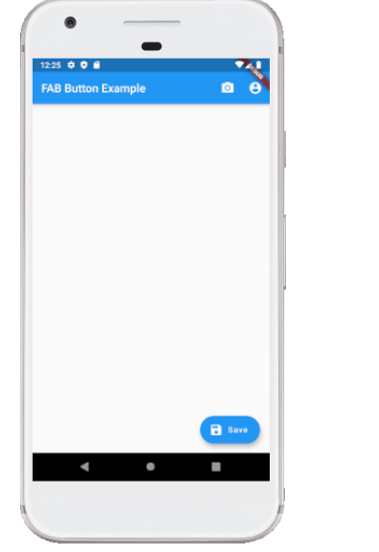
child: Icon(Icons.navigation),

backgroundColor: Colors.green,

foregroundColor: Colors.white,

onPressed: () => {},

),



**#12.4. Drop Down Button (Nút thả xuống)**

Một nút thả xuống được sử dụng để tạo một lớp phủ đẹp mắt trên màn hình cho phép người dùng chọn bất kỳ mục nào từ nhiều tùy chọn. Flutter cho phép một cách đơn giản để triển khai hộp thả xuống hoặc nút thả xuống. Nút này hiển thị mục hiện được chọn và một mũi tên mở menu để chọn một mục từ nhiều tùy chọn.

Flutter cung cấp một **widget DropdownButton** để triển khai một danh sách thả xuống. Chúng ta có thể đặt nó ở bất kỳ đâu trong ứng dụng của mình.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

child: DropdownButton(

value: \_itemSelected,

items: \_dropdownMenuItems,

onChanged: (value) {

setState(() {

\_itemSelected = value;

});

}),

**#12.5. Icon Button (Nút biếu tượng)**

IconButton là một **hình ảnh được in** trên widget Material. Nó là một widget hữu ích mang lại cho giao diện người dùng Flutter một cảm giác thiết kế material design. Chúng tôi cũng có thể tùy chỉnh giao diện của nút này. Nói một cách dễ hiểu, nó là một biểu tượng phản ứng khi người dùng chạm vào nó.

NoneBashCSSCC#ElixirErlangGoGraphQLGroovyHaskellHCLHTMLINIJavaJavaScriptJSONJSXKotlinLispLuaMermaid DiagramNixObjective-COCamlPerlPHPPowershellPythonRubyRustScalaSQLSoliditySwiftTOMLTSXTypeScriptVisual BasicYAMLZigCopy

IconButton(

icon: Icon(Icons.volume\_up),

iconSize: 50,

color: Colors.brown,

tooltip: 'Increase volume by 5',

onPressed: () {

setState(() {

\_speakervolume += 5;

});

},

),

