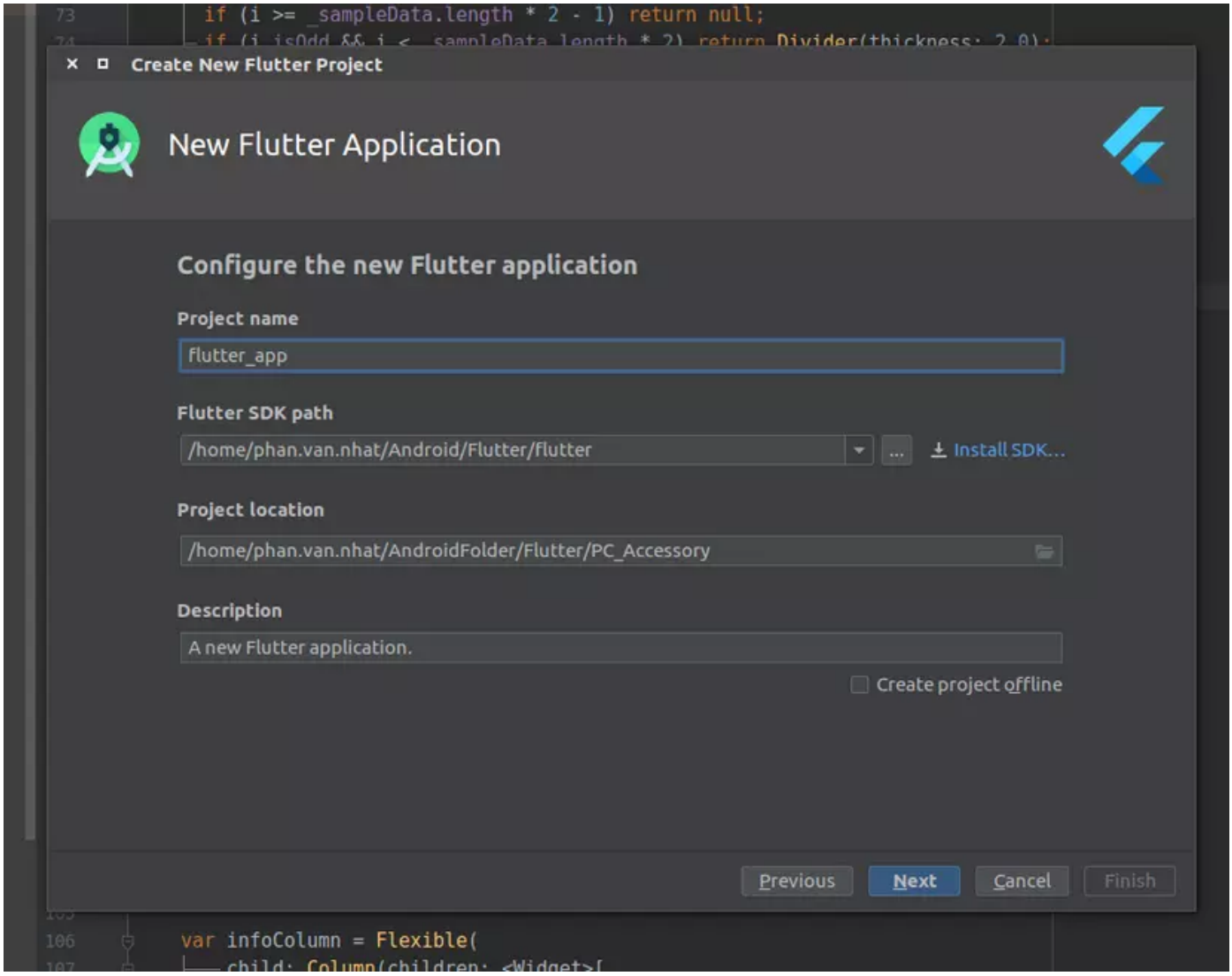
Create Flutter Project

1. Vào **File** -> **New** -> **New Flutter Project** (nó ở dòng thứ 2) -> Chọn cái đầu tiên là **Flutter Application** -> , chúng ta sẽ được một bảng tùy chọn như này :

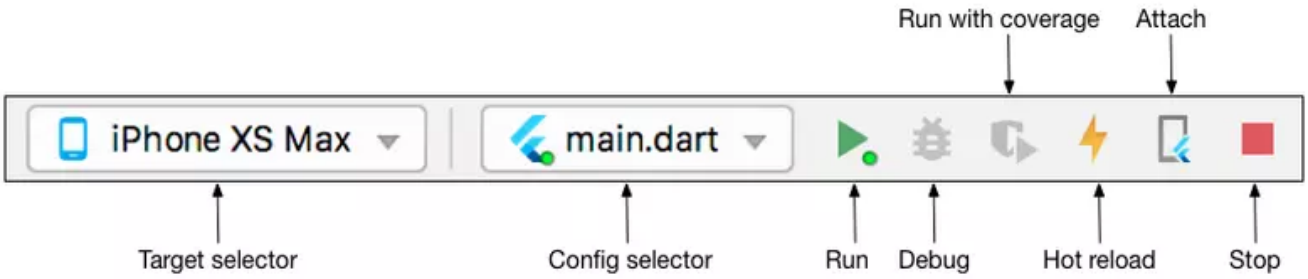


Cần phải lưu ý ở chỗ **Flutter SDK path** , các bạn nhớ úc nãy chúng ta đã download file SDK và giải nén nó. Đây là lúc cần dùng, hãy chọn icon [...] và trỏ tới SDK flutter mà chúng ta đã giải nén lúc nãy. Những field còn lại thì tự field theo ý mình thôi.

Và như vậy là đã set up xong, chúng ta cùng bắt đầu viết một project nào !

2. Build ứng dụng Flutter đầu tiên : Hello world

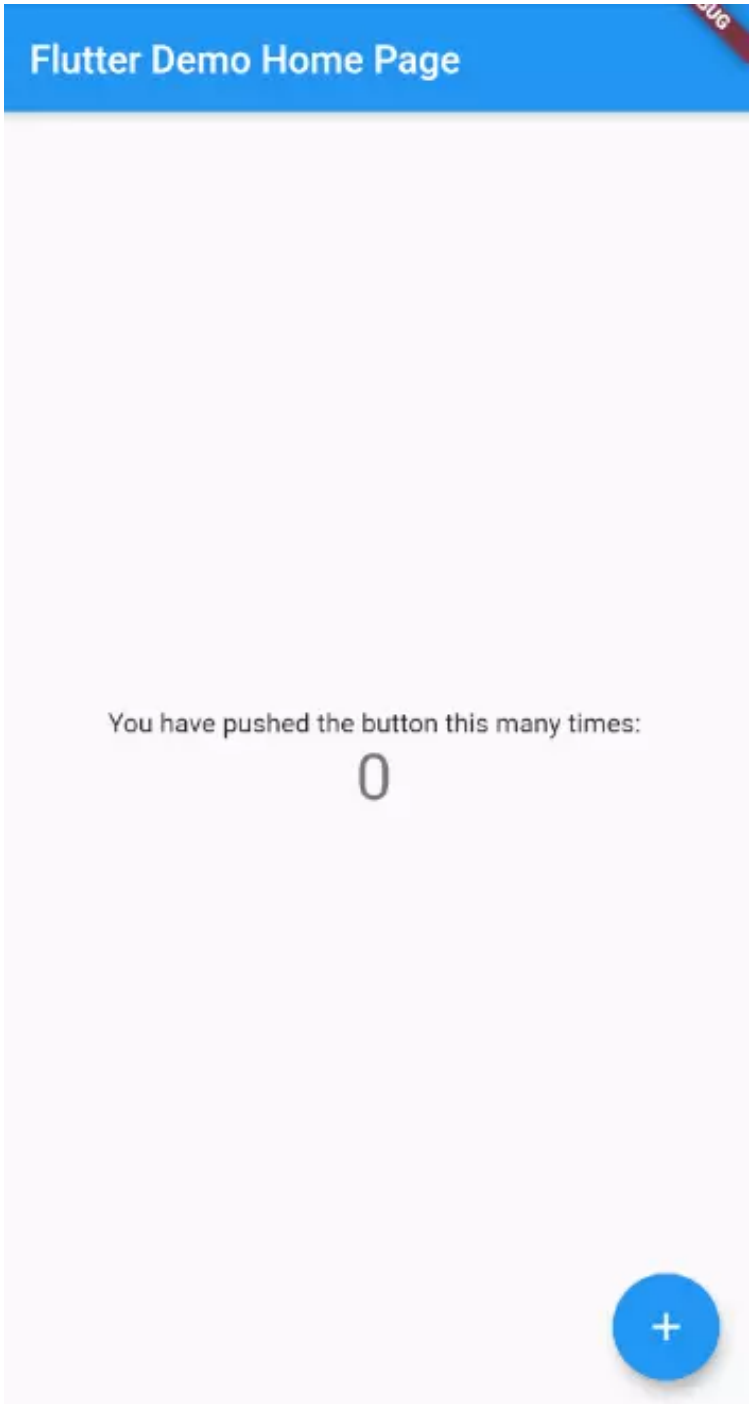
Sau khi hoàn thành xong mục 1, ở file main.dart, chúng ta sẽ thấy các line code sample của google, thôi thì tranh thủ chạy thử xem họ code những gì nào. Nhìn trên tool bar thì chúng ta sẽ thấy được một giao diện như sau:



Nếu đã quen với Android studio thì có thể nhanh chóng hiểu các tác dụng của những option này, từ trái qua phải lần lượt sẽ là các chức năng cần chú ý :

- Target selector: Chọn device để chạy
- Config selector: Chọn file chứa cofig code để chạy
- Run, Debug
- Hot reload: khi có chỉnh sửa bất cứ điều gì trong code, chúng ta có thể dùng Hot-reload để update những thay đổi mà không cần phải chạy là toàn bộ ứng dụng tốn thời gian, ngoài ra dùng Hot-reload thì những thao tác trên ứng dụng vẫn được dữ lại.
- Stop : Dừng ứng dụng đang chạy.

Ok. thử chạy code sample mà Google đã viết sẵn thôi, nhìn qua máy ảo chúng ta sẽ được một giao diện như này:

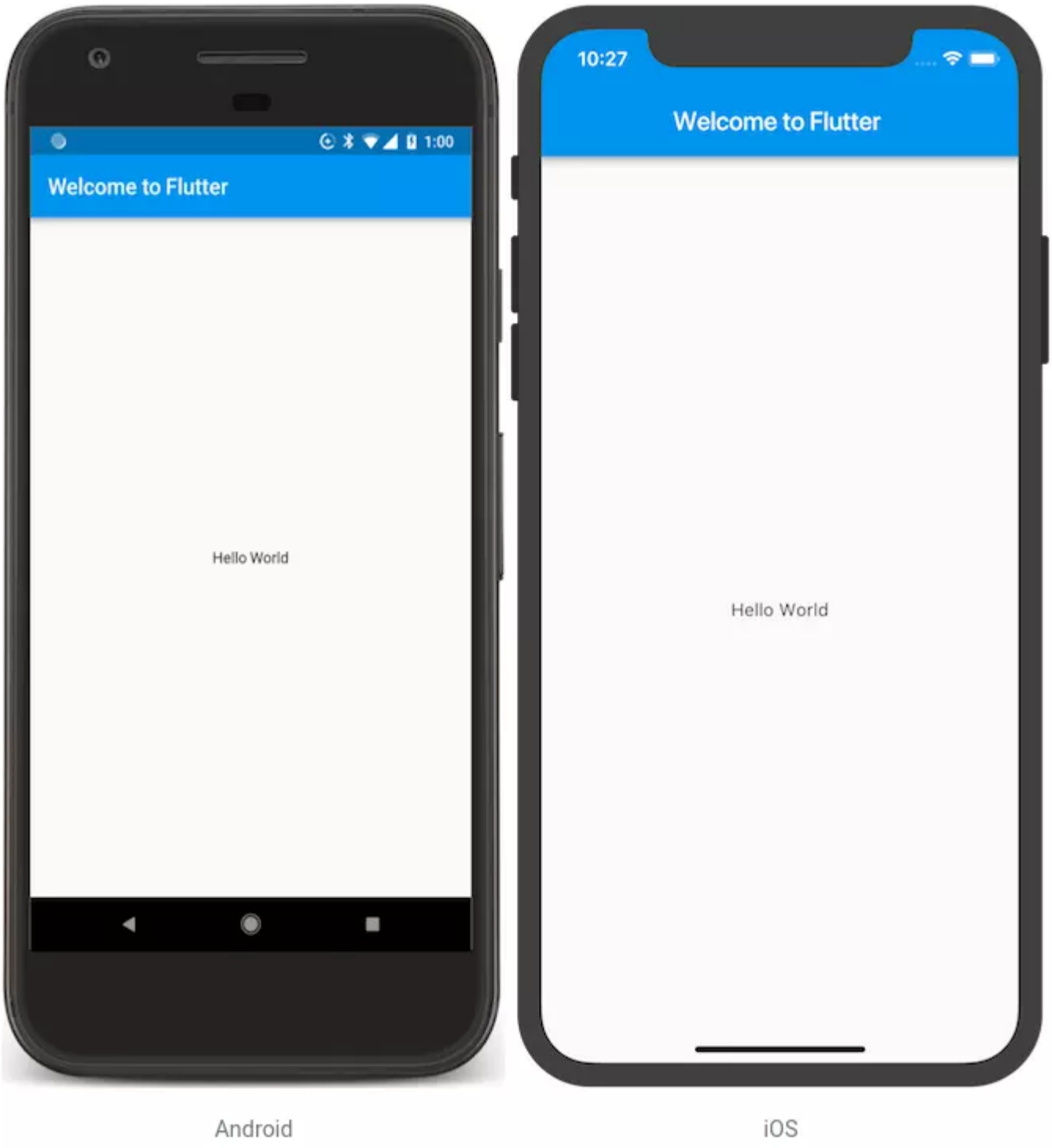


Nhìn sample code khá khó hiểu phải không? xóa nó đi và viết thử chương trình "Hello world" đã đi vào huyền thoại trước đã (yaoming). Xóa hết code ở main.dart và thay thế bằng đoạn code dưới đây.

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
```

```
          title: Text('Welcome to Flutter'),
        ),
        body: Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

Chạy lên và chúng ta sẽ được giao diện như thế này :



Phân tích :

- Đầu tiên, chúng ta cần phải làm quen với một khái niệm mới được sử dụng trong Flutter, đó là **Widget** . Ý tưởng chính của Widget là xây dựng lên những giao diện chúng ta nhìn thấy thông qua các trạng thái, cấu hình mà nó định nghĩa. Có vẻ hơi rối đúng không? Như chúng ta thấy ở trên đoạn code `child: Text('Hello World')` , `Text` ở đây chính là một Widget, trong này nó định nghĩa ra một string `'Hello World'` mà chúng ta có thể nhìn thấy. Như ở trong android native thì một Widget tương tự như là một View.

- method **main()** : Đây là nơi bắt đầu của ứng dụng, nơi thực thi tất cả những gì được tạo ra như method, biến, hàm ... Chắc các bạn cũng khá quen với method này rồi. Nó sử dụng ký hiệu arrow (=>). Sử dụng ký hiệu mũi tên cho các hàm hoặc phương thức một dòng. Sau dấu (=>) là những gì method main chạy và trả về .
- Trong example trên

```
MyApp
```

đang kế thừa từ

```
StatelessWidget
```

, biến bản thân MyApp thành một Widget. Trong Flutter thì hầu như tất cả mọi thứ đều là Widget, bao gồm cả alignment, padding, và layout.

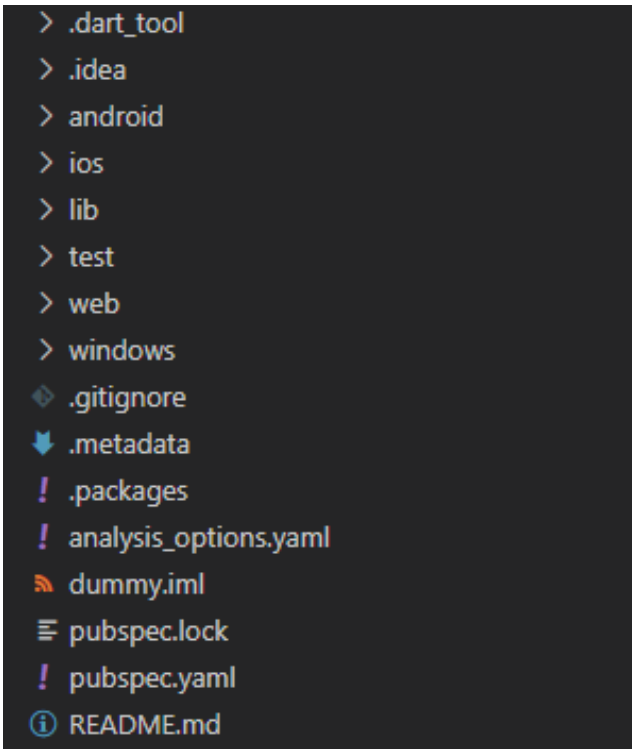
  - `StatelessWidget` là widget không có state. Nó không chấp nhận sự thay đổi bên trong nó. Còn đối với sự thay đổi từ bên ngoài (widget cha) thì nó sẽ thụ động thay đổi theo. Nó chỉ đơn thuần là nhận dữ liệu vào và hiển thị, chúng ta không thể thay đổi bất cứ điều gì, muốn tạo một variable cũng phải là final (ko được thay đổi). Bản thân nó cũng không có hàm createState mà thay vào đó là hàm build(BuildContext)
- `MaterialApp` là điểm khởi đầu của ứng dụng, nó cho Flutter biết rằng chúng ta sẽ sử dụng các thành phần Material và tuân theo thiết kế material design trong ứng dụng của mình. Các định nghĩa ở đây bao gồm màu sắc chủ đạo, dartThem, locale, navigation...
- `Scaffold` cho phép chúng ta triển khai các widget ứng dụng chuẩn material mà hầu hết các ứng dụng đều có. Chẳng hạn như AppBar, BottomAppBar, FloatingActionButton, BottomSheet, Drawer, Snackbar. Scaffold được thiết kế để trở thành vùng chứa cấp cao nhất cho MaterialApp mặc dù không cần thiết phải lồng một Scaffold.
- `Center` : như cái tên, nó đưa mọi thứ nằm bên trong nó vào giữa màn hình.
- `Text` : Hiển thị text, chúng ta có thể thêm slyle cho text này với thuộc tính `TextStyle` .
- Và cuối cùng khi lồng các Widget với nhau hãy chú ý cần phải khai báo đúng những params cho nó với các định nghĩa param như : `title:` , `body:` , ... tương ứng. Sẽ có lỗi compiler nếu thiếu những định nghĩa này.

Đây mới chỉ là những Widget căn bản. Còn rất nhiều Widget mà chúng ta sẽ học trong qua trình build một dự án, mình sẽ không liệt kê toàn bộ chúng mà sẽ giải thích qua những áp dụng thực tế vào code như trên.

** Tham khảo https://flutter.dev/docs

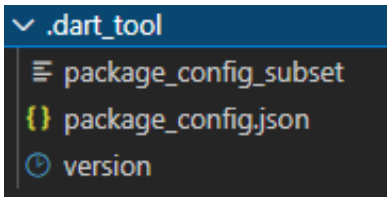
Hiểu về kiến trúc 1 dự án Flutter

Chúng ta sẽ bắt đầu với việc xem qua các thành phần có trong một dự án mặc định của Flutter khi ta tạo project.



Những thư mục chúng ta không cần quan tâm

.dart_tool folder



Thư mục này chứa các tệp được sử dụng bởi các công cụ Dart.

.metadata file

Tệp được kiểm soát bởi công cụ Flutter và không nên được chỉnh sửa theo cách thủ công.

.packages file

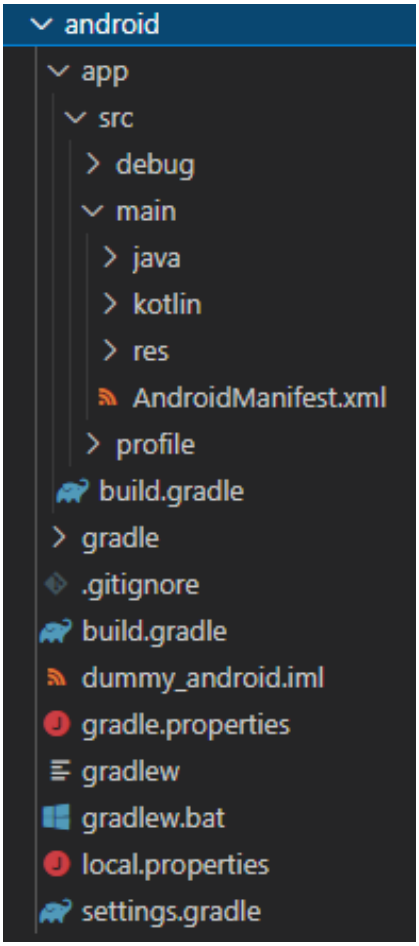
File này chứa thông tin về các gói được sử dụng trong dự án. Chúng ta không thể làm gì với tệp này. Từ sau Dart 2.17 thì chúng ta có thể gỡ bỏ file này. Thường mình sẽ xóa file này để clear cache, tránh việc đã pull lib mới về rồi nhưng project vẫn nhận lib cũ.

dummy.iml

Tệp này được quản lý bởi Flutter SDK và bạn không nên chỉnh sửa nó theo cách thủ công. Tên luôn khớp với tên dự án của bạn trong quá trình tạo ban đầu.

Những folder, file mà chúng ta cần quan tâm

android folder



Thư mục này chứa tất cả dữ liệu có liên quan để biên dịch và tạo một ứng dụng Android đang hoạt động (apk hoặc appbundle). Có hai tệp mà bạn có nhiều khả năng sẽ cần chỉnh sửa trong quá trình phát triển:

`build.gradle` trong `android / app` Nó chứa `applicationId` được yêu cầu khi gửi lên Google Play Store. Đảm bảo rằng giá trị này khớp với giá trị bạn chỉ định trong Google Play Console trước khi tải ứng dụng của bạn lên.
Ngoài ra cũng có một tệp `build.gradle` khác trong thư mục con `android`.

`AndroidManifest.xml` trong `android / app / src / main` Tại đây bạn có thể chỉ định ý định hoặc khả năng mà ứng dụng của bạn sẽ sử dụng. Khi sử dụng các gói của bên thứ 3, hướng dẫn cài đặt của họ thường bao gồm thông tin chi tiết về những gì cần được thêm vào đây. Ví dụ `permission`

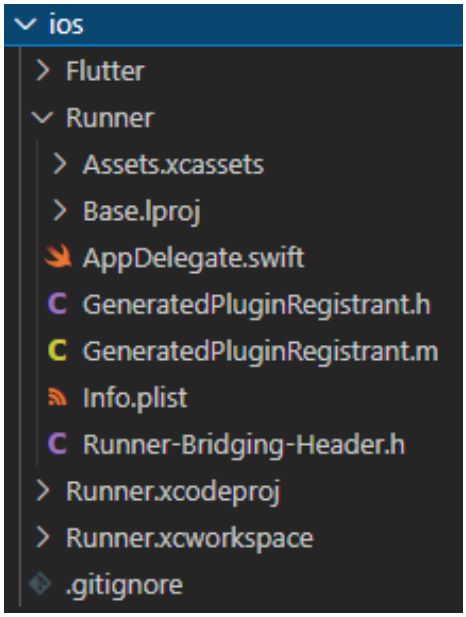
Ngoài ra nếu chúng ta cần triển khai `methodChannel` thì chúng ta sẽ viết ở dưới tầng Android này để tạo giao thức trao đổi thông tin giữa tầng Flutter và native.

build folder

Thư mục `build` sẽ được tạo khi Flutter được thực thi lần đầu tiên. Nó chứa các tệp được tạo cần thiết để chạy ứng dụng trên các nền tảng khác nhau. Mỗi nền tảng có thư mục con riêng của nó.

Tại đây bạn cũng có thể tìm thấy file `apk` của mình khi bạn run dự án.

ios folder

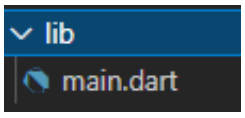


Tương tự với thư mục android nhưng dành cho các tệp liên quan đến ios. Các tệp bạn có nhiều khả năng sẽ chỉnh sửa là:

`AppDelegate.swift` trong `ios / Runner` Điểm vào ứng dụng ios. Các gói của bên thứ 3 có thể cần phải thêm một số logic khởi tạo ở đây. Bạn cũng có thể kiểm tra tài liệu hướng dẫn tích hợp của họ để biết cách triển khai ở đây

`Info.plist` trong `ios / Runner` Chứa tất cả các cài đặt liên quan đến ứng dụng. Các tính năng và khả năng bổ sung có thể yêu cầu mục nhập mới trong tệp này.

lib folder



Thư mục lib là nơi chứa mã ứng dụng của bạn. Và chúng ta sẽ làm việc với anh bạn `lib` này nhiều. Chúng ta sẽ triển khai mã nguồn của dự án tại đây.

test folder

Đây là thư mực chứa các file để chúng ta viết test. Các file test ở đây đều có hậu tố trong tên là `_test.dart`.

.gitignore file

Chứa thông tin về những tệp và thư mục nào sẽ được loại trừ khỏi kiểm tra phiên bản với git. Ngoài ra, bạn có thể muốn thêm bất kỳ khóa API nào mà ứng dụng của bạn sẽ sử dụng để chúng không bị lộ trong kho lưu trữ công khai.

analysis_options.yaml file

Ở đây chúng ta sẽ có thể thêm hoặc loại bỏ một số analysis trong dự án của mình. Bạn có thể xem kĩ hơn tại đây

pubspec.yaml

Pubspec chứa mô tả dự án, khai báo `assets`, `font`, cách `libs` bên thứ 3 mà bận muốn sử dụng. Tất cả các gói bên thứ 3 của bạn sẽ được xác định ở đây để các công cụ Flutter biết phiên bản nào và tải nó về. Bạn có thể tìm thêm thông tin về pubspec.yaml trong bài viết dưới đây.

main.dart

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

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Hello World Flutter Application',
      theme: ThemeData(
        // This is the theme of your application.
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'Home page'),
    );
  }
}

class MyHomePage extends StatelessWidget {
  MyHomePage({Key key, this.title}) : super(key: key);
  // This widget is the home page of your application.
  final String title;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
```

```
29          title: Text(this.title),
30        ),
31        body: Center(
32          child: Text('Hello World - Vietcombank'),
33        ),
34      );
35    }
36  }
```

Để bắt đầu với việc thiết kế giao diện, trước tiên bạn cần import một số thư viện liên quan tới giao diện. Ở đây, chúng ta đã import một **Material package** . Package này cho phép bạn tạo giao diện người dùng theo ngôn ngữ thiết kế Material design do Android chỉ định.

Dòng thứ hai là một điểm chạy đầu tiên của các ứng dụng Flutter tương tự như phương thức main trong các ngôn ngữ lập trình khác. Nó gọi hàm **runApp** và chuyển cho nó một đối tượng của **MyApp** Mục đích chính của hàm này là để bạn chạy màn hình nào đó đầu tiên.

Dòng 5 đến 18 là một `widget` được sử dụng để tạo giao diện người dùng. Ở đây, `StatelessWidget` không duy trì bất kỳ trạng thái nào của widget. Class `MyApp` kế thừa `StatelessWidget` và ghi đè phương thức `build` của nó Phương pháp `build` được sử dụng để tạo một phần giao diện người dùng của ứng dụng. Trong khối lệnh này, phương thức `build` sử dụng `MaterialApp`, một widget cung cấp cho lập trình viên một bộ khung trong quá trình phát triển ứng dụng như `title`, `theme`, `home`,... Mỗi ứng dụng Flutter chỉ nên chứa 1 `MaterialApp`.

Dòng 20-36, class `MyHomePage` cũng tương tự như `MyApp`, ngoại trừ nó sẽ được return về một widget là `Scaffold`. Scaffold Widget cũng là một widget được sử dụng nhiều nhất. Nó cung cấp cho lập trình viên một bộ khung chứa sẵn các thuộc tính quan trong để xây dựng giao diện 1 màn hình như `appBar`, `body`,... . `AppBar` hiển thị tiêu đề của ứng dụng và thuộc tính `body` hiển thị nội dung thực của ứng dụng. Ở đây dễ thấy, `appBar` đang được định danh là một đối tượng `AppBar` có thuộc tính `title` chính là một `Text` widget. Tương tự với thuộc tính `body` đang được truyền vào là một `Center` widget có thuộc tính `child` là một `Text` widget.

Giới thiệu về runApp và Material App

runApp()

Một ứng dụng Flutter đơn giản bằng cách gọi `runApp()` với 1 widget:

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

void main() {
  runApp(
    Center(
      child: Text(
        'Hello, world!',
        textDirection: TextDirection.ltr,
      ),
    ),
  );
}
```

`runApp()` lấy widget phía trên và lấy nó là root của widget tree. Trong ví dụ này, widget tree bao gồm 2 widgets , Center widget và con của nó là Text widget. Khi chạy ứng dụng lên bạn sẽ thấy `Hello, world!` sẽ nằm giữa màn hình. Hướng của Text cần được chỉ định trong trường hợp này; Khi mà chúng ta sử dụng `MaterialApp` widget, nó sẽ xử lý giúp chúng ta.

Material App

– 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.

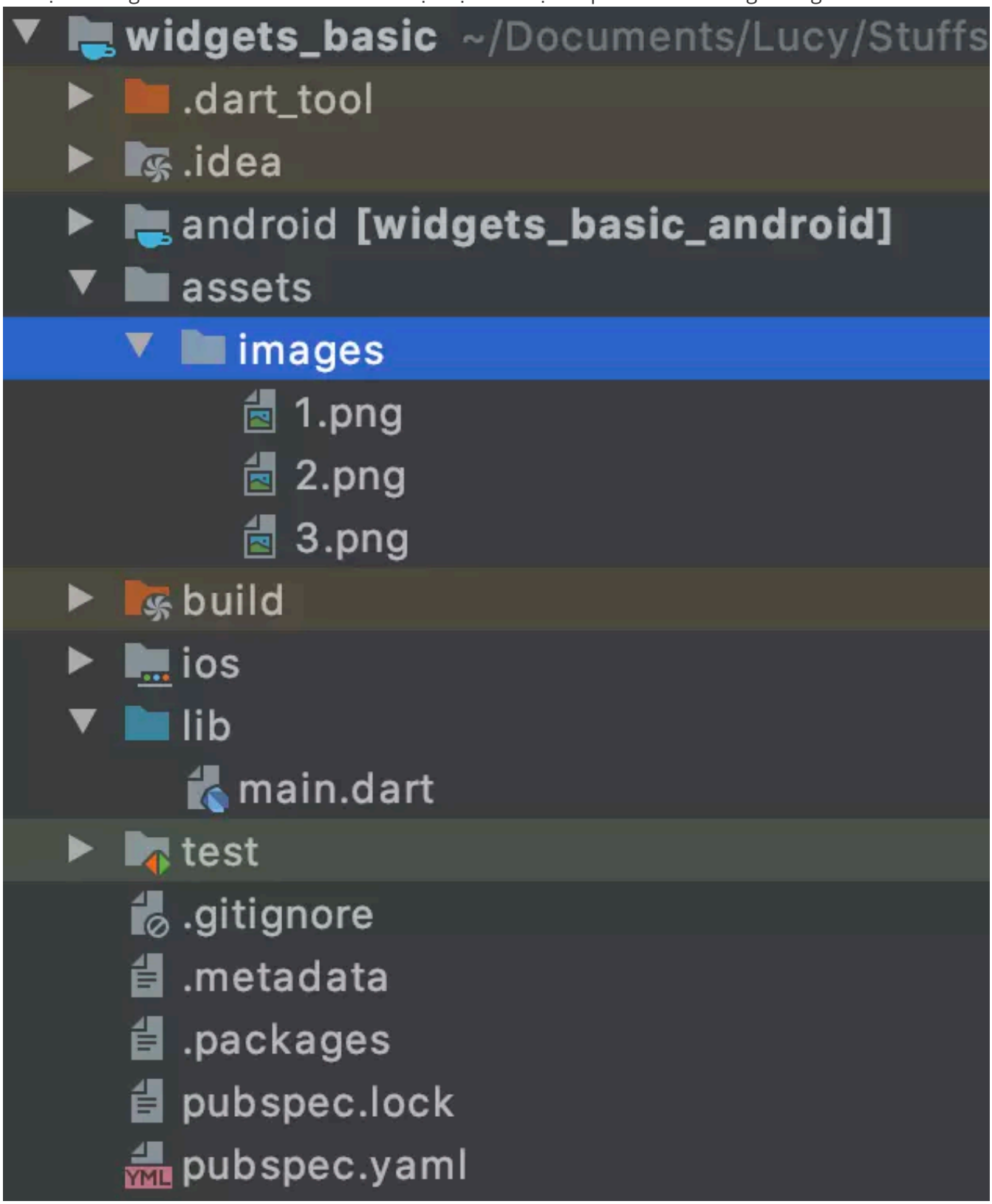
```dart
 @override
  Widget build(BuildContext context) {

    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(),
    );
  }
```

Ngoài ra, còn có các thuộc tính khác trong MaterialApp – Xem thêm.

Cách sử dụng asets trong Flutter

Tạo folder chứa ảnh

Tạo mới một folder chứa ảnh ngang hàng với file pubspec.yaml, bạn có thể đặt ảnh bên trong folder images rồi đặt ở trong assets như hình bên dưới hoặc đặt ảnh trực tiếp vào assets cũng không sao.



Mỗi image sẽ được định danh bằng một đường dẫn cụ thể tới nơi mà image file được đặt. Thứ tự sắp xếp của image không quan trọng, và tên của folder chứa ảnh cũng không quan trọng, bạn có thể đặt tuỳ ý. Trong quá trình build, Flutter sẽ đặt các image này vào một kho lưu trữ đặc biệt được gọi là asset bundle, nơi mà app sẽ đọc dữ liệu ở runtime.

Copy ảnh vào folder vừa tạo

Bây giờ, bạn có thể đưa ảnh vào folder images, ví dụ như ở trên, mình đã đưa vào ba ảnh 1.png, 2.png, 3.png. Lúc này đường dẫn đến ảnh sẽ như sau:

```
assets/images/1.png
```

Đăng ký folder ảnh vào file pubspec.yaml

Để sử dụng được ảnh từ assets, ta cần đăng ký nó vào file pubspec.yaml, bạn có thể tìm thấy nó ngay trong cây thư mục root của dự án. Ngay bên dưới dòng `uses-material-design: true`, khai báo assets cho những image mà ta định dùng như sau:

```
assets:
    - assets/images/1.png
    - assets/images/2.png
    - assets/images/3.png
```

Hoặc nếu bạn muốn load tất cả các ảnh trong folder này, chỉ cần khai báo như sau là được:

```
assets:
    - assets/images/
```

Sử dụng ảnh trong code

Để load ảnh vào code, ta sử dụng cú pháp sau:

```
Image.asset('assets/images/1.png')
```

Hãy cùng đặt ba ảnh được khai báo ở trên vào một Row nhé.

```
body: Center(
          child: Row(
            children: [
              Image.asset('assets/images/1.png'),
              Image.asset('assets/images/2.png'),
              Image.asset('assets/images/3.png'),
            ],
      ),
),
```

Chạy app lên, bạn sẽ thấy ảnh số một sẽ được hiển thị trên màn hình sau:

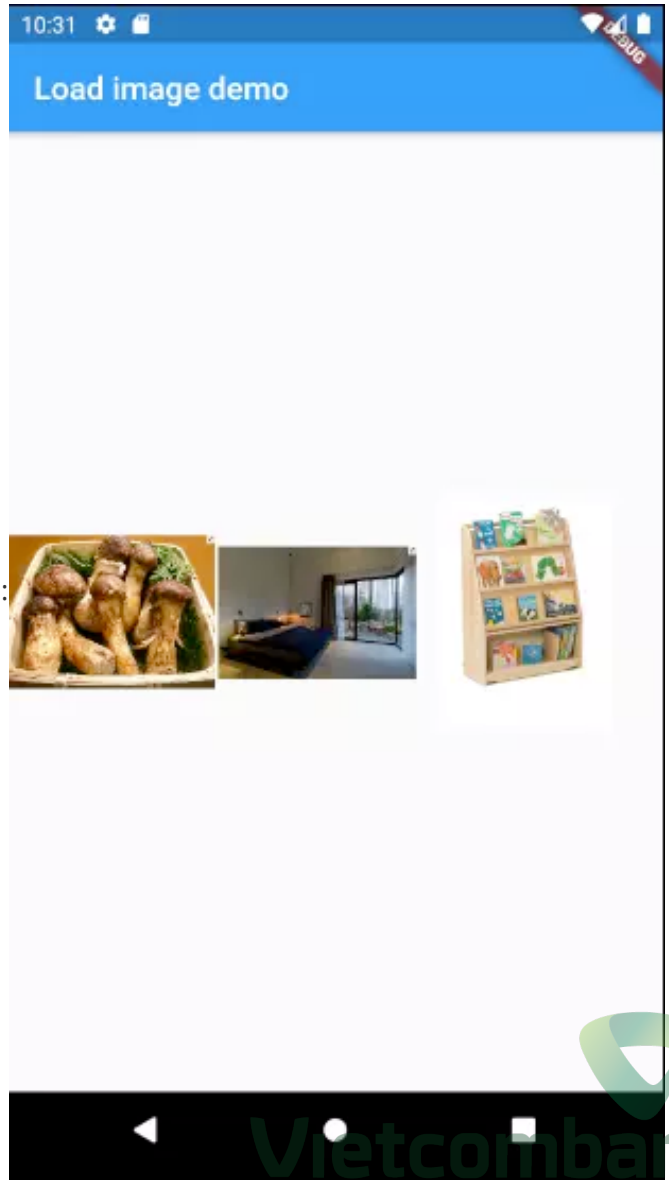
Nhưng hãy khoan, ảnh số 2 và 3 đi đâu rồi??? Nếu nhìn kỹ, bạn sẽ thấy một đường đứng màu vàng sọc chạy dọc theo góc phải bức ảnh, đường sọc này ám chỉ bức ảnh có width lớn hơn màn hình hiển thị nên nó không hiện hết được ảnh.

Ngoài ra, bạn có nhận ra đây là nấm Matsutake, một loại nấm siêu ngon của Nhật Bản không, hãy cùng chỉnh lại để giỏ nấm siêu đắt đỏ này hiển thị đầy đủ nhé 😛.

Để set kích thước cho ảnh, ta có thể thêm vào width và height cho từng ảnh:

```
body: Center(
          child: Row(
            children: [
              Image.asset('assets/images/1.png', width: 130, height: 150,),
              Image.asset('assets/images/2.png', width: 130, height: 150,),
              Image.asset('assets/images/3.png', width: 130, height: 150,),
            ],
        ),
),
```

Bây giờ thì cả ba ảnh đều được lên hình rồi nhé:



Tuy nhiên, khoảng cách giữa cách ảnh vẫn chưa phân bổ đều, để dàn đều ảnh, cách tốt nhất là đặt các bức ảnh vào Expanded, và set .spaceEvenly cho mainAxisAlignment để chia đều các ảnh.

```
body: Center(
          child: Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: [
              Expanded(
                child: Image.asset('assets/images/1.png'),
              ),
              Expanded(
                child: Image.asset('assets/images/2.png'),
              ),
              Expanded(
                child: Image.asset('assets/images/3.png'),
              ),
        ],
    ),
 ),
```

Ngoài ra, bạn cũng có thể set thêm các thuộc tính khác để chỉnh UI, ví dụ như thêm flex vào để set độ scale của ảnh...

Đến đây, chắc hẳn các bạn đã tự mình load được ảnh vào một dự án Flutter rồi, rất nhanh và tiện phải không nào ^.^

** Thảm khảo tài liệu tại viblo.asia

Statefull và Stateless - Cách tạo Statefull và Stateless Widget

Stateful vs **Stateless Widget**

Trong bài viết này, chúng ta sẽ tìm hiểu chi tiết về các state trong **Flutter**: bao gồm **Stateful** và **Stateless** widget.

State trong Flutter là gì?

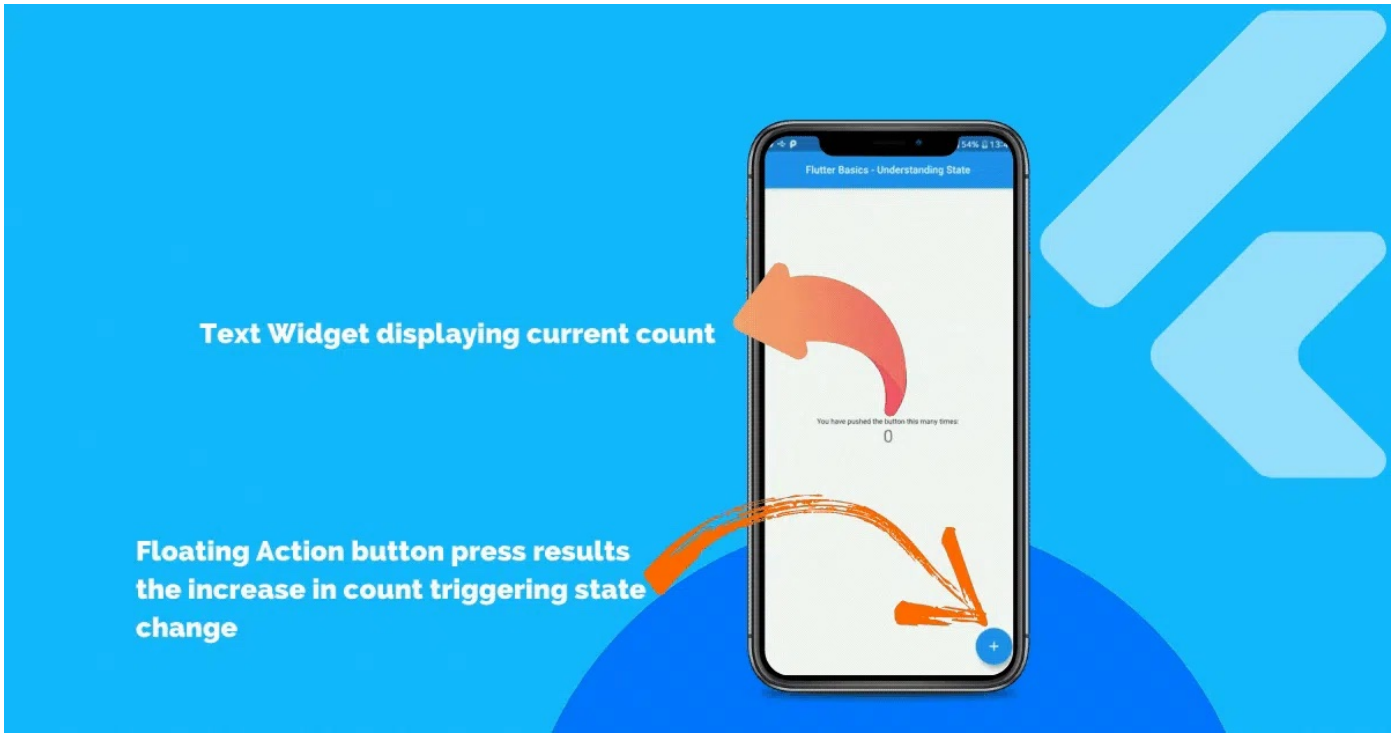
Mình chắc chắn rằng bạn đã gặp nhiều định nghĩa về **state** trên internet trước đây và thành thật mà nói tất cả chúng dường như hơi phức tạp để hiểu vì các định nghĩa đó khá sách vở. Thực sự là bản thân mình cũng đã từng như thế.

Vì thế mình sẽ cố gắng để giải thích nó đơn giản hết sức có thể về State trong Flutter:

> **State** là một số dữ liệu hoặc thông tin được ứng dụng của bạn sử dụng. Nó 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.

Về cơ bản, **Flutter** lưu giữ snapshot của widget hiện đang được hiển thị và nếu bất kỳ dữ liệu nào bên trong widget đó thay đổi thì dữ liệu snapshot trước đó và dữ liệu hiện tại sẽ được so sánh và **widget** liên quan sẽ được **rebuild**!

Về lý thuyết State là vậy. Sau đây, mình sẽ sử dụng state trong một app ví dụ thực tế. Hãy xem xét ví dụ về ứng dụng mặc định mà Flutter cung cấp cho chúng ta khi chúng ta khởi tạo dự án mới.



Bạn có thể thấy rằng khi nhấn **Floating Action Button**, số đếm hiển thị trong **Text Widget** phản ánh thay đổi. Bạn có thể thắc mắc điều này xảy ra như thế nào? Vì vậy chúng ta hãy nghiên cứu thêm.

- Giá trị ban đầu của biến counter là `0`, do đó `0` được hiển thị ngay từ lúc đầu.
- Ngay sau khi nút được nhấn, một hàm được gắn vào phương thức `onClick(){...}` của nút làm tăng giá trị của bộ đếm lên `1`. Vì vậy, giá trị tăng `1` trên mỗi lần nhấn nút.
- Ngay sau khi giá trị của các biến đếm thay đổi, nó sẽ phát hiện ra những thay đổi này, do đó sẽ kích hoạt phương thức `build(){...}` của widget mà đếm biến hoạt động.
- Khi phương thức `build(){...}` được kích hoạt, một bản build hoàn chỉnh của tất cả các child/nested widgets bên trong widget đó sẽ được rebuild với dữ liệu mới.
- Do đó, chúng ta thấy dữ liệu được cập nhật trong thời gian thực. Đây là cách hoạt động của **state** trong **Flutter**.

> Trong trường hợp bạn đang thắc mắc về cách **rebuild** giao diện người dùng hoàn chỉnh hiệu quả như thế nào, hãy yên tâm vì **Flutter** khá hiệu quả và thông minh trong việc phát hiện các **widget** cần **rebuild** và chỉ **rebuild** chúng.

App-Wide State

Các giá trị nếu người dùng được xác thực, một số dữ liệu được tìm nạp từ backend/server có thể được coi là app-wide state. Các loại dữ liệu này kiểm soát tổng thể cả ứng dụng.

Widget state

Widget state có thể được coi là một cái gì đó giống như:

- Vòng quay loading hiển thị khi dữ liệu đang được tìm kiếm từ backend/server.
- Giá trị input vào của người dùng hiện tại hoặc số lần người dùng đã nhấn vào nút trong trường hợp ứng dụng mẫu của chúng ta.

Widget states có thể và sẽ thay đổi thường xuyên nhất trong mọi trường hợp.

Bây giờ chúng ta đã hiểu rõ về những gì state làm và cách các widget được rebuild, chúng ta hãy tiếp tục tìm hiểu về các **Stateless widget** và **Stateful widget**.

Stateless Widgets

Như tên của nó, tất cả các widget không thể/sẽ không tự rebuild ngay cả khi dữ liệu hoặc các biến bên trong chúng thay đổi. Một stateless widget điển hình trông như thế này:

DART

copy

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

class DummyWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      // your nested widgets and children
      child: ...
    );
  }
}
```

Vì vậy, bất kỳ widget nào kế thừa `StatelessWidget` class từ material package được coi là một stateless widget bởi Flutter. Các widget này sẽ không thay đổi khi người dùng tương tác với chúng, ngay cả khi dữ liệu bên trong chúng thay đổi. Nó chỉ quan tâm đến việc hiển thị một số dữ liệu nhất định với một style nhất định.

- Các widget này chỉ được tạo một lần duy nhất, tức là khi chúng được hiển thị trên màn hình, chúng sẽ không thay đổi cho đến khi và 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 các thay đổi về hàm dựng (constructor) của chúng.
- Hãy xem xét trường hợp một stateful widget là parent của một stateless widget. Nếu phương thức build(){...} của stateful widget gốc được kích hoạt bằng cách nào đó thì child stateless widget cũng được rebuild.
- Stateless widget sẽ rebuild nếu dữ liệu bên ngoài chúng thay đổi nếu **Provider** được đính kèm với stateless widget và widget đó là consumer hoặc active listener đối với provider. Ngay sau khi các giá trị của provider thay đổi, stateless widget sẽ rebuild.
- Một số ví dụ về stateless widgets là `Text()`, `Column()`, `Row()`, v.v.

Bây giờ, sẽ đến các **stateful widget**.

Stateful Widgets

Tất cả các widget kế thừa `StatefulWidget` class được coi là các stateful widget. Các widget này sẽ kích hoạt các phương thức build của chúng ngay khi dữ liệu bên trong chúng thay đổi hoặc dữ liệu bên ngoài được cung cấp cho chúng thông qua các thay đổi về hàm dựng của chúng. Trong cả hai trường hợp, phương thức `build(){...}` của các widget này được kích hoạt. Hãy xem nhanh một ví dụ về **stateful widget**:

DART

copy

```dart
class DummyWidget extends StatefulWidget {
  @override
  _DummyWidgetState createState() => _DummyWidgetState();
}

class _DummyWidgetState extends State<DummyWidget> {
  bool _isGreen = false;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: _isGreen ? Colors.green : Colors.red,
      appBar: AppBar(
        title: Text('Your First App'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            FlatButton(
              onPressed: () {
                setState(() {
                  _isGreen = !_isGreen;
                });
              },
              child: Text(_isGreen ? 'TURN RED' : 'TURN GREEN'),
            ),
          ],
        ),
      ),
    );
  }
}
```

Ví dụ trên là từ bài viết trước của mình. Chúng ta có thể thấy rõ rằng có rất nhiều thứ đang diễn ra ở **stateful widget** so với **stateless widget**.

Điều gì tạo nên một Stateful Widget?

Các stateful widget không chỉ là một class mà là sự kết hợp của hai class.

```
class DummyWidget extends StatefulWidget {
  @override
  _DummyWidgetState createState() => _DummyWidgetState();
}
```

Class thứ nhất kế thừa `StatefulWidget` và override phương thức `createState()`. Phương thức `createState()` được khai báo bởi `StatefulWidget` class.

> Chúng ta sử dụng tường minh phương thức `@override` để cho Flutter biết rằng chúng ta đang trả về một đối tượng mới dựa trên class thứ hai và chúng ta có thể kết nối cả hai class này.

Class thứ hai bao gồm tất cả logic liên quan đến widget state.

```
class _DummyWidgetState extends State<DummyWidget> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
        // rest of the code
    );
  }
}
```

Trong class này được đặt tên là `_<widget_name>State` trong đó `_` xác định rằng nó là private và tên widget bắt buộc để cho biết widget's state mà nó sẽ nắm giữ.

`State` là một class được import từ `material package`. Vì vậy, chúng ta cung cấp cho nó tên của widget để cho phép người dùng biết widget's state mà chúng ta muốn liên kết với nó.

> Ở class đầu tiên có thể được tạo lại ngay khi dữ liệu bên ngoài được cung cấp thông qua phương thức khởi tạo thay đổi. Nhưng vì chúng ta cần giữ lại state bên trong của widget khi việc rebuild do thay đổi từ Widget cha. Do đó, chúng ta cần hai class, một cái để kích hoạt build, nhận vào các dữ liệu từ bên ngoài và một cái để kích hoạt rebuild cho chính bản thân nó.

`setState(){...}` method

Trong ví dụ trên, chúng ta đang sử dụng phương thức `setState(){...}` được thực thi sau khi người dùng nhấn `FlatButton`. Hàm này được cung cấp bởi `State class` mà chúng ta kế thừa từ material package.

Chúng ta bọc tất cả logic/code bên trong hàm này để thay đổi dữ liệu. Dữ liệu nội bộ 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.

Kết

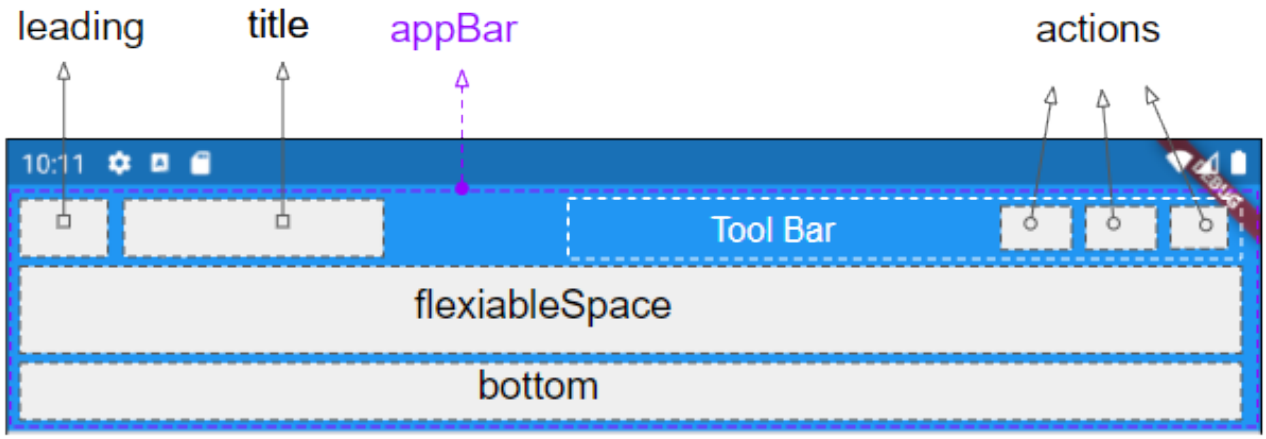
Đó là những kiến thức chuyên sâu về cách các **stateless** và **stateful** widget hoạt động. Mình hy vọng rằng bây giờ bạn đã hiểu rõ ràng hơn nhiều về cách thức hoạt động của những thứ này. Trong hầu hết các ứng dụng, chúng ta sử dụng các stateless widget thường xuyên hơn so với các stateful widget bởi vì trong hầu hết các trường hợp, tất cả những gì chúng ta quan tâm là hiển thị dữ liệu mà thôi.

Nhưng đôi khi chúng ta cũng sử dụng **stateful** widgets để phục vụ cho việc cập nhật, thay đổi từ trong chính bản thân Widget này. Trong đó có thể là: thay đổi trạng thái khi người dùng tương tác hoặc nhận các sự kiện khác trong ứng dụng.

Bài viết được lược dịch từ Shashank Biplav.

Tạo AppBar

Trong **Flutter**, **AppBar** (Thanh ứng dụng) bao gồm một thanh công cụ (Tool Bar) và các **Widget** tiềm năng khác. Cụ thể, **AppBar** được chia làm 5 khu vực **leading, title, Tool Bar (actions), flexiableSpace, bottom**.



AppBar Constructor :

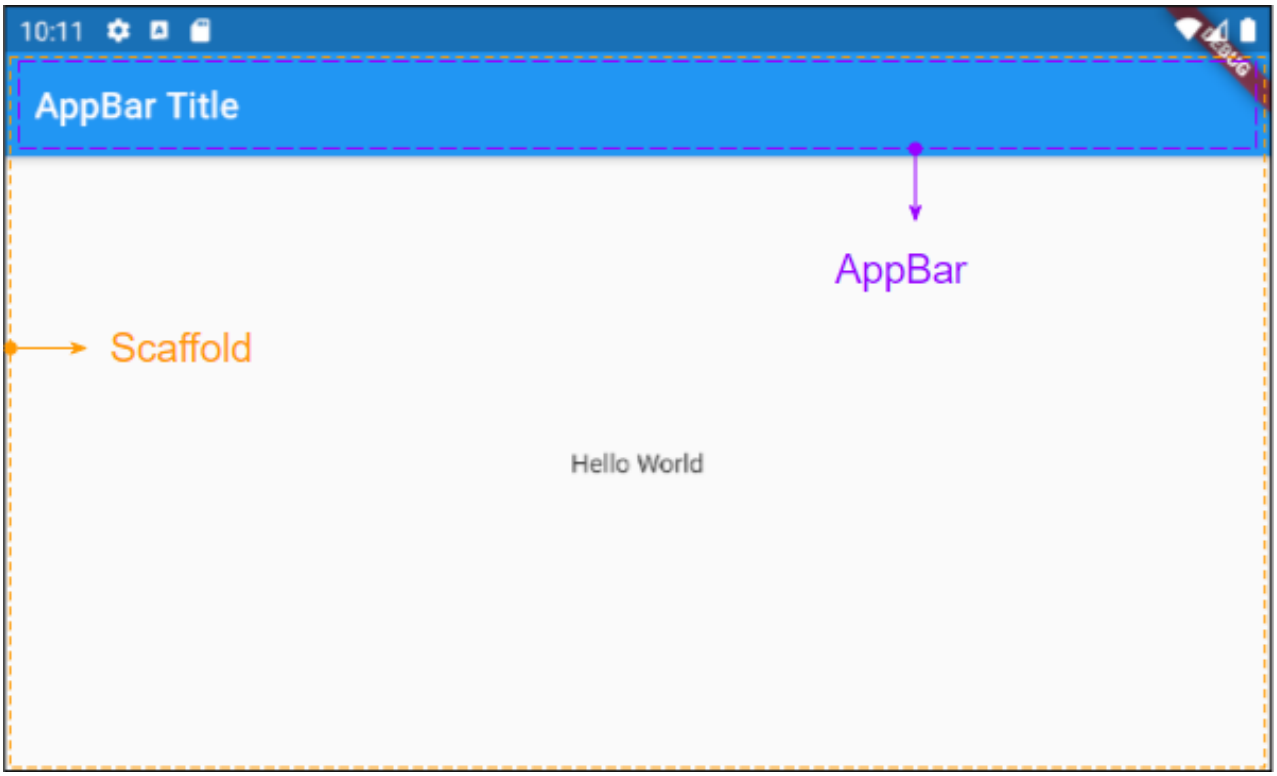
AppBar Constructor

Bạn đang đọc: Hướng dẫn và ví dụ Flutter AppBar

```
AppBar( {Key key,
    Widget leading,
    bool automaticallyImplyLeading: true,
    Widget title,
    List actions,
    Widget flexibleSpace,
    PreferredSizeWidget bottom,
    double elevation,
    Color shadowColor,
    ShapeBorder shape,
    Color backgroundColor,
    Brightness brightness,
    IconThemeData iconTheme,
    IconThemeData actionsIconTheme,
    TextTheme textTheme,
```

```
    bool primary: true,
    bool centerTitle,
    bool excludeHeaderSemantics: false,
    double titleSpacing: NavigationToolbar.kMiddleSpacing,
    double toolbarOpacity: 1.0,
    double bottomOpacity: 1.0,
    double toolbarHeight
  }
)
```

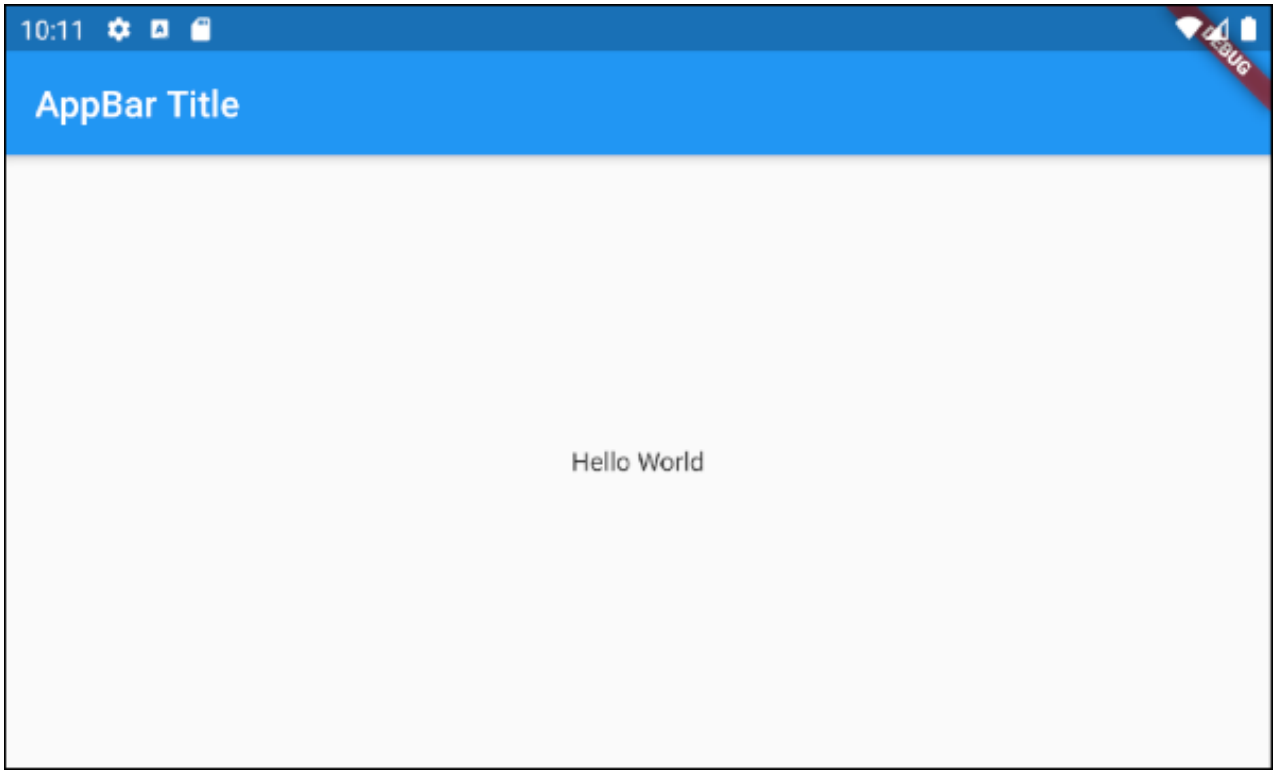
AppBar thường được đặt trong một **Scaffold** (Khung) thông qua property **Scaffold.appBar**. **AppBar** sẽ có chiều cao cố định và xuất hiện phía trên (top) của **Scaffbar**. Nếu bạn muốn có một thanh ứng dụng có thể cuộn hãy sử dụng **SliverAppBar**.



2- title

```
Widget title;
```

Ví dụ một **AppBar** đơn giản chỉ bao gồm một tiêu đề, và được đặt trong một **Scaffold**. Nó sẽ xuất hiện phía trên (top) của **Scaffold**.

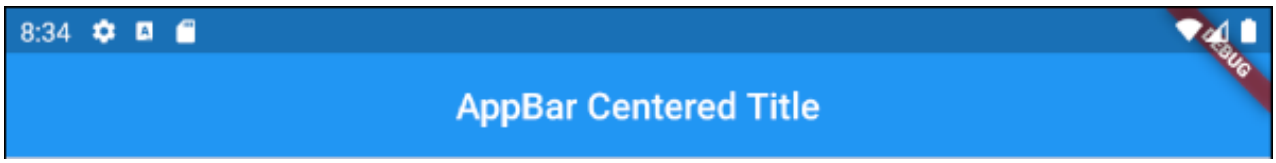


main.dart (title ex1)

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

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Title of Application',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      home: MyHomePage(),
    );
  }
}

class MyHomePage extends StatelessWidget {
  MyHomePage({Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
```

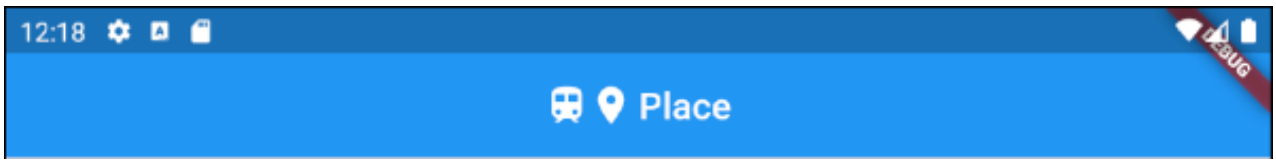
```
        title: Text("AppBar Title"),
      ),
      body: Center(
          child:  Text(
            'Hello World',
          )
      ),
    );
  }
}
```

Ví dụ một **AppBar** với tiêu đề được căn giữa hoặc căn phải:




```
AppBar(
    title: Align (
        child: Text("AppBar Centered Title"),
        alignment: Alignment.center
    )
);

AppBar(
    title: Align (
        child: Text("AppBar Right Title"),
        alignment: Alignment.centerRight
    )
);

AppBar(
    title:  Text("AppBar Centered Title"),
    centerTitle: true,
);
```

Ví dụ tạo một **"Title Widget"** bao gồm các biểu tượng (icon) và văn bản.



main.dart (title ex3)

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

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

```
6
7  class MyApp extends StatelessWidget {
8
9    @override
10   Widget build(BuildContext context) {
11     return MaterialApp(
12       title: 'Title of Application',
13       theme: ThemeData(
14         primarySwatch: Colors.blue,
15         visualDensity: VisualDensity.adaptivePlatformDensity,
16       ),
17       home: MyHomePage(),
18     );
19   }
20 }
21
22 class MyHomePage extends StatelessWidget {
23   MyHomePage({Key key}) : super(key: key);
24
25   @override
26   Widget build(BuildContext context) {
27     return Scaffold(
28       appBar: AppBar(
29           title: IconTitleWidget()
30       ),
31       body: Center(
32           child: Text(
33             'Flutter AppBar Tutorial',
34           )
35       ),
36     );
37   }
38 }
39
40 class IconTitleWidget extends StatelessWidget {
41
42   @override
43   Widget build(BuildContext context) {
44     imageCache.clear();
45     return Row (
46       mainAxisAlignment: MainAxisAlignment.center, // Centers horizontally
47       crossAxisAlignment: CrossAxisAlignment.center, // Centers vertically
48       children: [
49         Icon(Icons.train),
50         Icon(Icons.place),
51         // The SizedBox provides an immediate spacing between the widgets
52         SizedBox (
53           width: 3,
54         ),
```

```
        Text(
          "Place",
        )
      ],
    );
  }
}
```

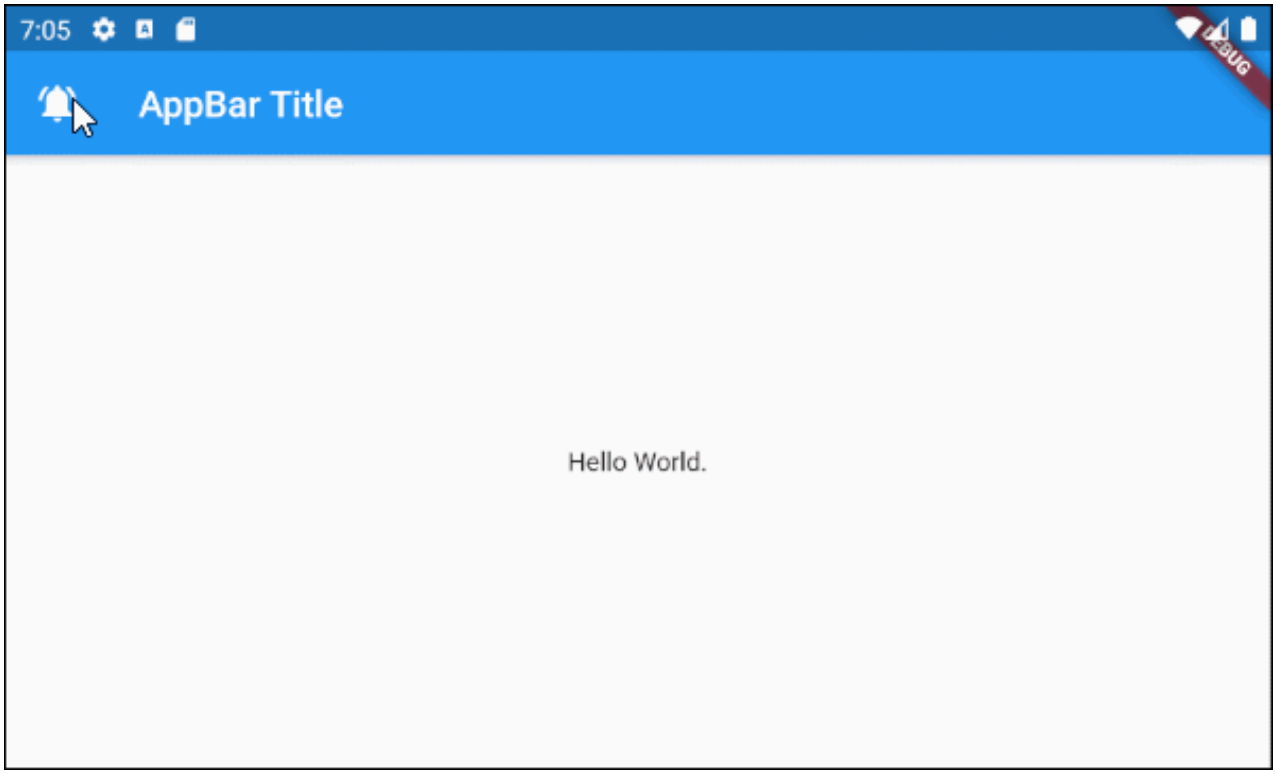
3- leading

leading là một **Widget** được đặt phía trước vùng **title**, thông thường nó là một **Icon** hoặc **IconButton**.

```
Widget leading;
```

Ví dụ: **leading** là một **IconButton**, khi người dùng nhấp vào **IconButton** một hành động nào đó sẽ được thực thi.

```
// Example: leading is an IconButton
appBar: AppBar(
    title: Text("AppBar Title"),
    leading: IconButton(
        icon: Icon(Icons.notifications_active),
        onPressed: () {
             // Do something.
        }
    )
)

// Example: leading is an Icon
appBar: AppBar(
    title: Text("AppBar Title"),
    leading: Icon(Icons.notifications_active)
)
```

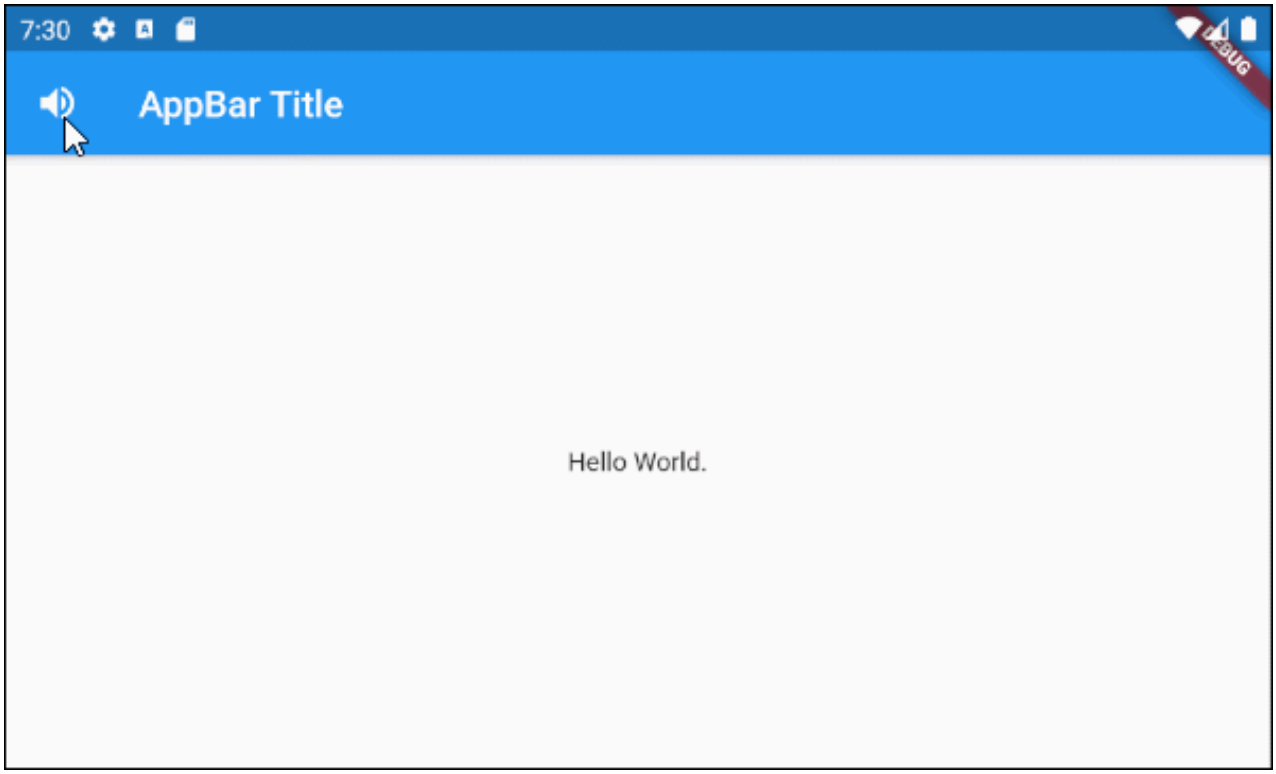


main.dart (leading ex1)

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

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Title of Application',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      home: MyHomePage(),
    );
  }
}

class MyHomePage extends StatelessWidget {
  MyHomePage({Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
```

```
          title: Text("AppBar Title"),
          leading: IconButton(
              icon: Icon(Icons.notifications_active),
              onPressed: () {
                showAlert(context);
              }
          )
      ),
      body: Center(
          child:  Text("Hello World.")
      ),
    );
  }

  void showAlert(BuildContext context) {
    showDialog(
        context: context,
        builder: (context) => AlertDialog(
          content: Text("Hi"),
        ));
  }
}
```

Ví dụ :



main.dart (leading ex2)

Xem thêm: Đặt tên miền Blog cá nhân như thế nào?

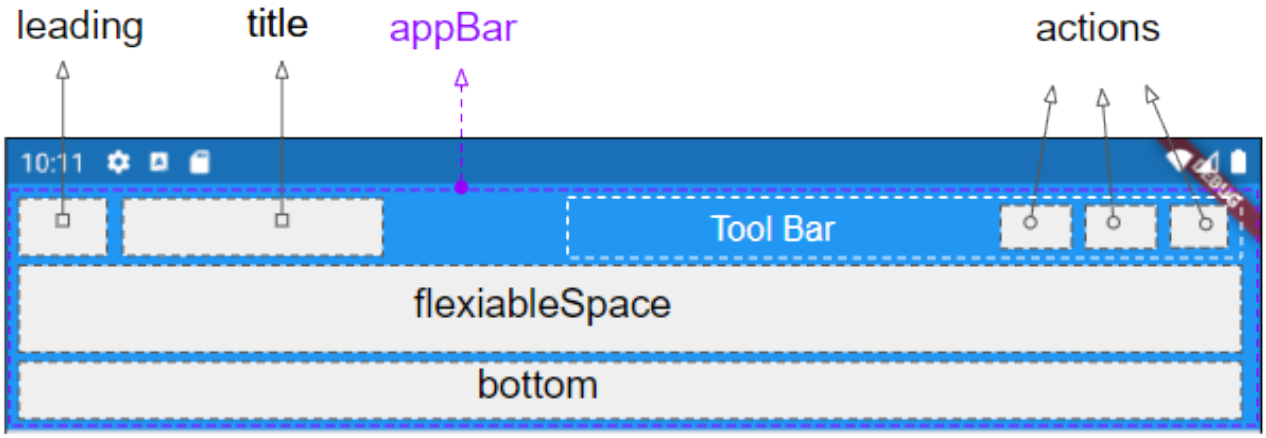

```
import 'package:flutter/material.dart';

```

```dart
void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Title of Application',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      home: MyHomePage(),
    );
  }
}

class MyHomePage extends StatelessWidget {
  MyHomePage({Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
          title: Text("AppBar Title"),
          leading: MyVolumeButton()
      ),
      body: Center(
          child:  Text("Hello World.")
      ),
    );
  }

}

class MyVolumeButton extends StatefulWidget {
  MyVolumeButton({Key key}) : super(key: key);

  @override
  State createState() {
    return MyVolumeButtonState();
  }
}

class MyVolumeButtonState extends State {
  bool volumeOn = true;

```

```
  @override
  Widget build(BuildContext context) {
    return IconButton(
        icon: this.volumeOn? Icon(Icons.volume_up):Icon(Icons.volume_mute),
        onPressed: () {
          // Set new State
          setState(() => this.volumeOn = !this.volumeOn);
        }
    );
  }
}
```

4- automaticallyImplyLeading

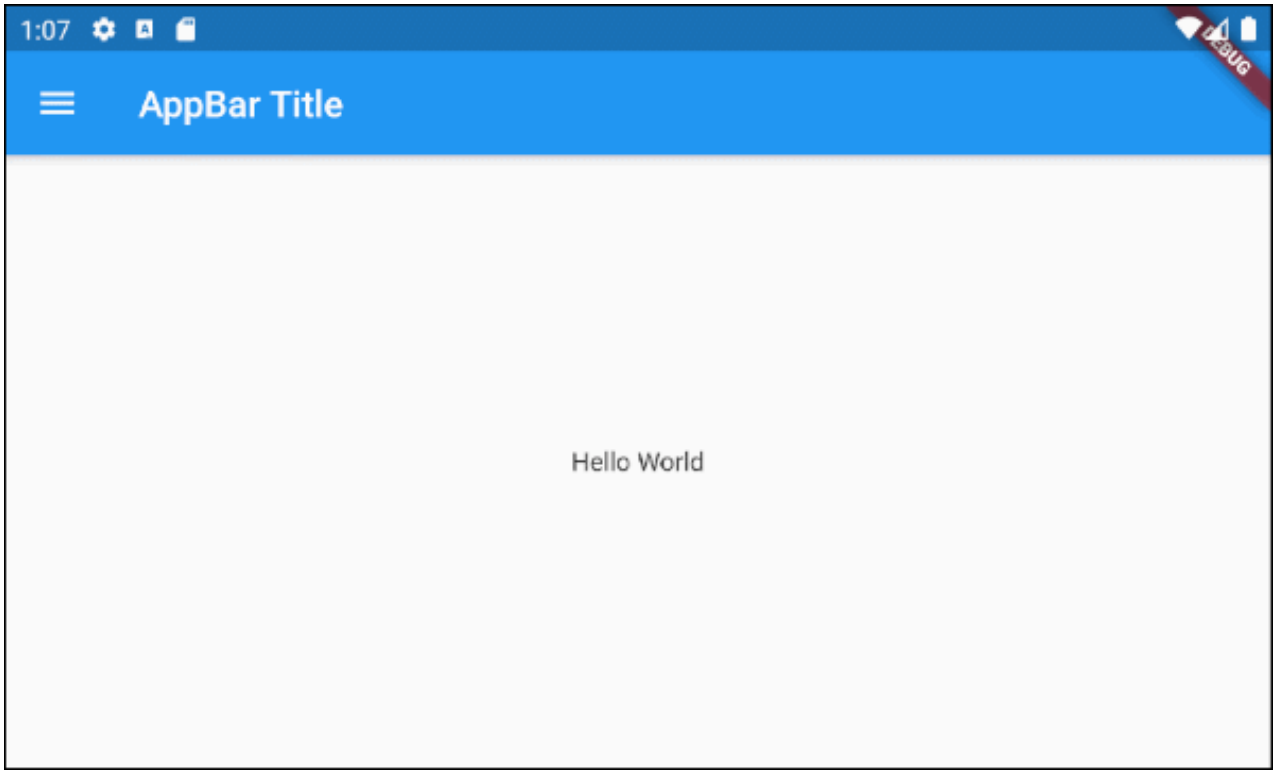
```
bool automaticallyImplyLeading: true
```



automaticallyImplyLeading là một property tùy chọn của **AppBar**, nó có giá trị mặc định là **true**. Khi bạn không đặt bất kỳ một **Widget** nào vào vùng **leading**, thì một **Widget** phù hợp có thể sẽ được tự động đặt vào đó, tùy theo ngữ cảnh.

Trương hợp 1: Một **IconButton** tự động được thêm vào vùng **leading** của **AppBar** để hỗ trợ mở ra một **Drawer** (Ngăn kéo) nếu các điều kiện sau đúng:

1. Vùng **leading** của **AppBar** rỗng.
2. **AppBar. automaticallyImplyLeading : true**.
3. **AppBar** được đặt trong một **Scaffold**.
4. **Scaffold** có chứa một **Drawer** (**Scaffold.drawer** được chỉ định).



Trường hợp 2: Một **IconButton – "BACK"** sẽ tự động được thêm vào vùng **leading** của một **AppBar** để hỗ trợ bạn quay lại màn hình trước đó nếu các điều kiện sau đây đúng:

1. Vùng **leading** của **AppBar** rỗng.
2. **AppBar. automaticallyImplyLeading : true**.
3. **AppBar.drawer : null**
4. Bạn đã nhẩy tới màn hình (screen) hiện tại từ một màn hình trước đó.

main.dart (automaticallyImplyLeading : true)

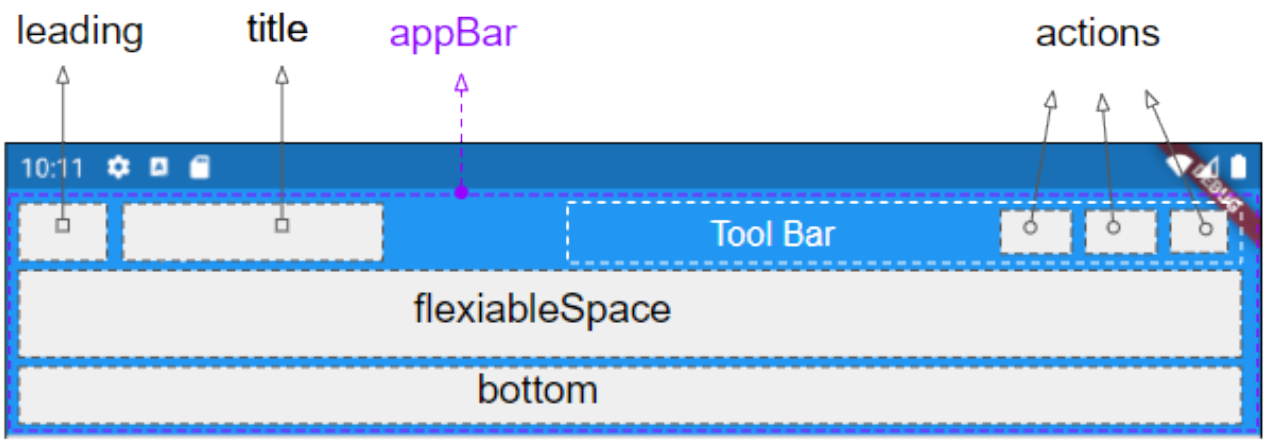
```dart
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Title of Application',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
```

```
      home: MyHomePage(),
    );
  }
}

class MyHomePage extends StatelessWidget {
  MyHomePage({Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      // AppBar with automaticallyImplyLeading = "true" (Default)
      appBar: AppBar(
          title: Text("AppBar Title"),
          automaticallyImplyLeading: true
      ),
      body: Center(
          child:  Text("Hello World.")
      ),
      drawer: Drawer(
        child: ListView(
          children: const  [
            DrawerHeader(
              decoration: BoxDecoration(
                color: Colors.green,
              ),
              child: Text(
                'My Drawer',
                style: TextStyle(
                  color: Colors.green,
                  fontSize: 24,
                ),
              ),
            ),
            ListTile(
              title: Text('Gallery'),
            ),
            ListTile(
              title: Text('Slideshow'),
            ),
          ],
        ),
      ),
    );
  }
}
```

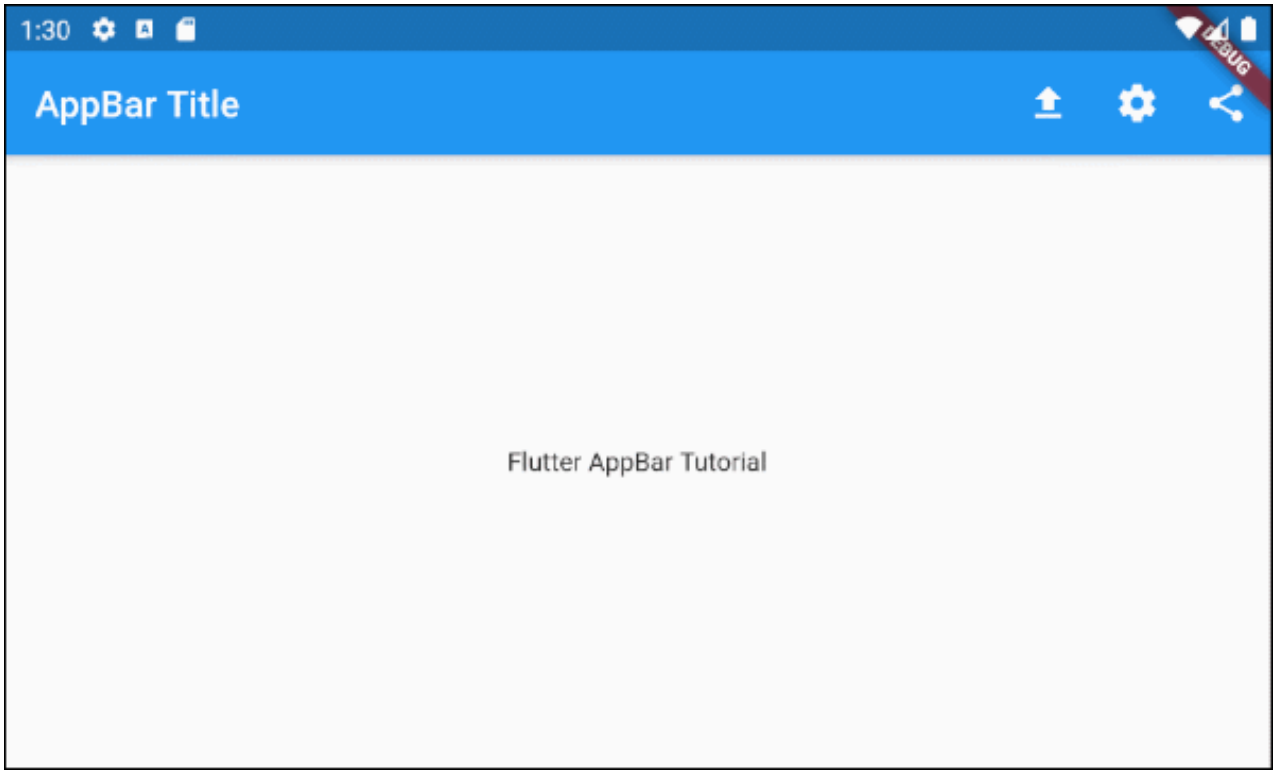
5- actions

Property **actions** cho phép bạn thêm các **action** (hành động) vào thanh công cụ (Tool bar) của **AppBar**. Thông thường **IconButton** sẽ được sử dụng cho mỗi **action** thông dụng, với **action** ít thông dụng hơn bạn hãy cân nhắc sử dụng **PopupMenuButton**.

```
List actions
```



Ví dụ thêm các **action** vào thanh công cụ của **AppBar**.



main.dart (actions ex1)

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

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

class MyApp extends StatelessWidget {
```

```
8
9   @override
10  Widget build(BuildContext context) {
11    return MaterialApp(
12      title: 'Title of Application',
13      theme: ThemeData(
14        primarySwatch: Colors.blue,
15        visualDensity: VisualDensity.adaptivePlatformDensity,
16      ),
17      home: MyHomePage(),
18    );
19  }
20 }
21
22 class MyHomePage extends StatelessWidget {
23   MyHomePage({Key key}) : super(key: key);
24
25   @override
26   Widget build(BuildContext context) {
27     return Scaffold(
28       appBar: AppBar(
29           title: Text("AppBar Title"),
30           actions:  [
31             IconButton(
32               icon: Icon(Icons.file_upload),
33               onPressed: () => {
34                 print("Click on upload button")
35               },
36             ),
37             IconButton(
38                 icon: Icon(Icons.settings),
39                 onPressed: () => {
40                   print("Click on settings button")
41                 }
42             ),
43             PopupMenuButton(
44               icon: Icon(Icons.share),
45               itemBuilder: (context) => [
46                 PopupMenuItem(
47                   value: 1,
48                   child: Text("Facebook"),
49
50                 ),
51                 PopupMenuItem(
52                   value: 2,
53                   child: Text("Instagram"),
54                 ),
55               ],
56             )
```

```
            ]
        ),
        body: Center(
            child: Text(
              'Flutter AppBar Tutorial',
            )
        ),
      );
  }
}
```

Chú ý: Chiều cao của các **action** bị giới hạn bởi chiều cao của thanh công cụ (Tool Bar), tuy nhiên bạn có thể sét đặt chiều cao của thanh công cụ thông qua property **toolbarHeight**.

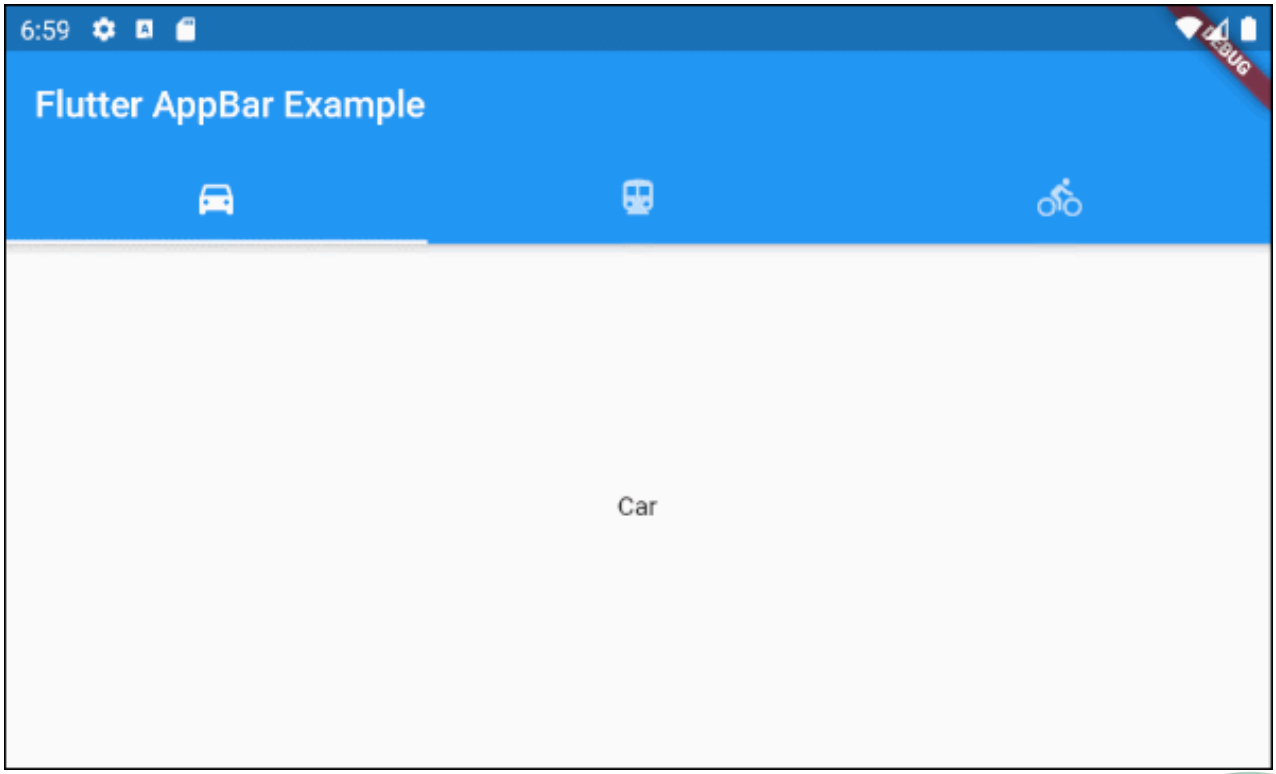
- TODO Link ?

6- bottom

Xem thêm: Cách đưa Blog cá nhân lên Instagram ➡️ Điểm dừng sáng tạo ▷ ➡️

Vùng **bottom** của **AppBar** thường được sử dụng để chứa một **TabBar**.

```
PreferredSizeWidget bottom;
```



main.dart (bottom ex1)

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

void main() {
```

```
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Title of Application',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      home: MyHomePage(),
    );
  }
}

class MyHomePage extends StatelessWidget {
  MyHomePage({Key key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return DefaultTabController(
        length: 3,
        child: Scaffold(
          appBar: AppBar(
            bottom: TabBar(
              tabs: [
                Tab(icon: Icon(Icons.directions_car)),
                Tab(icon: Icon(Icons.directions_transit)),
                Tab(icon: Icon(Icons.directions_bike)),
              ],
            ),
            title: Text('Flutter AppBar Example'),
          ),
          body: TabBarView (
            children: [
              Center(child: Text("Car")),
              Center(child: Text("Transit")),
              Center(child: Text("Bike"))
            ],
          ),
        )
    );
  }
}
```

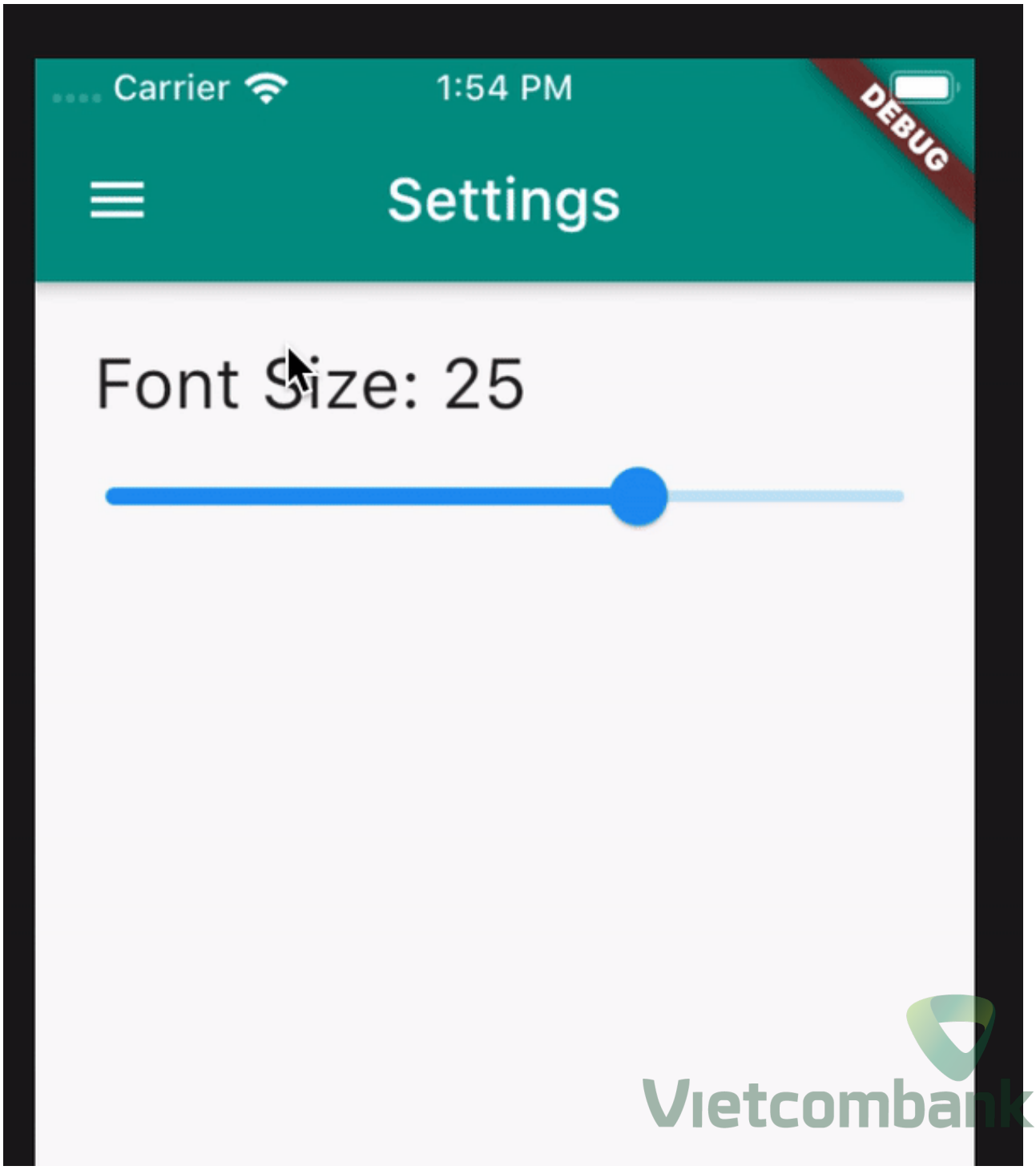
** Tham khảo tại leading10.vn

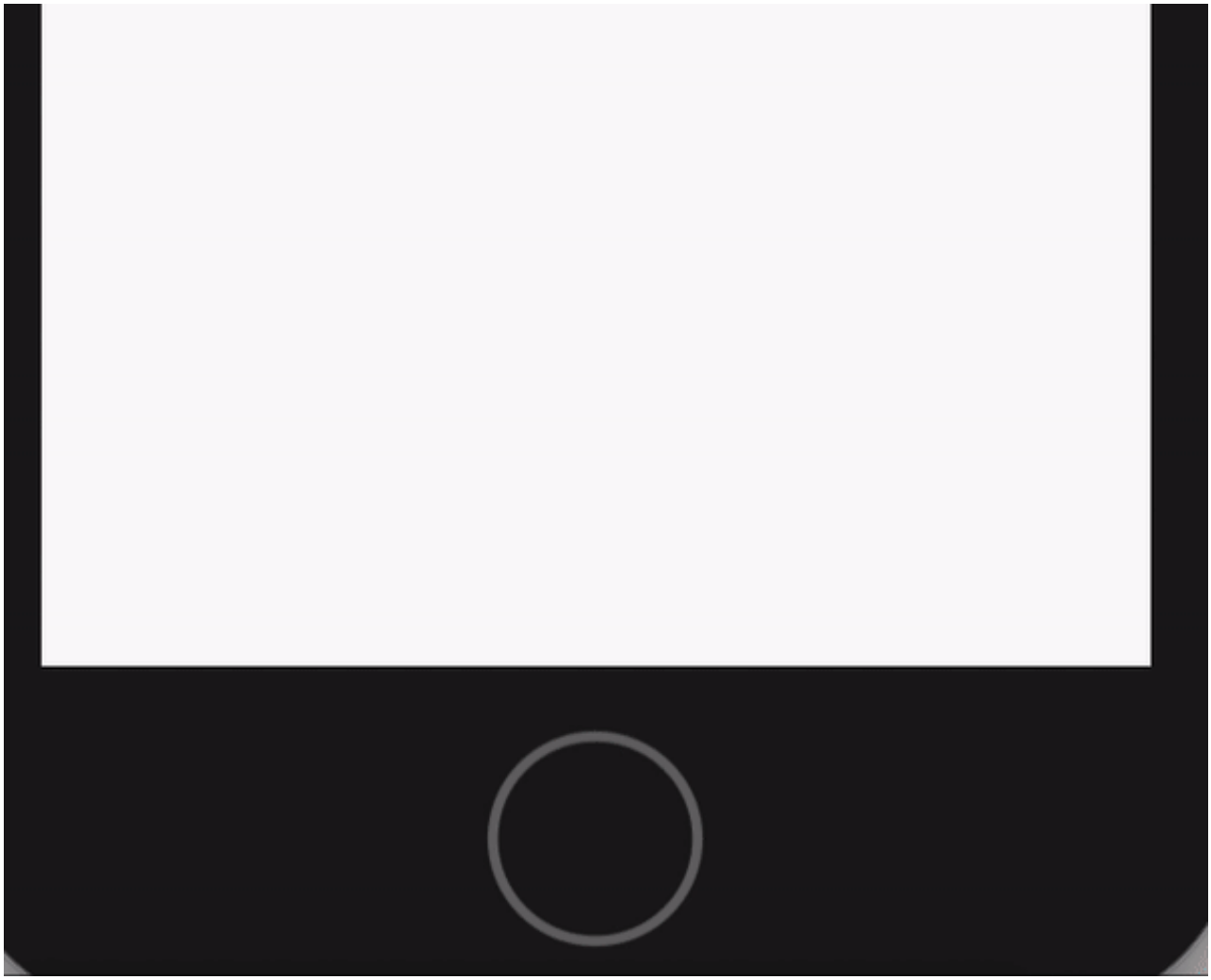
Quản lí state với Provider

Trong bài hướng dẫn này, tôi sẽ hướng dẫn bạn apply provider trong app. Demo app bao gồm 3 màn hình :

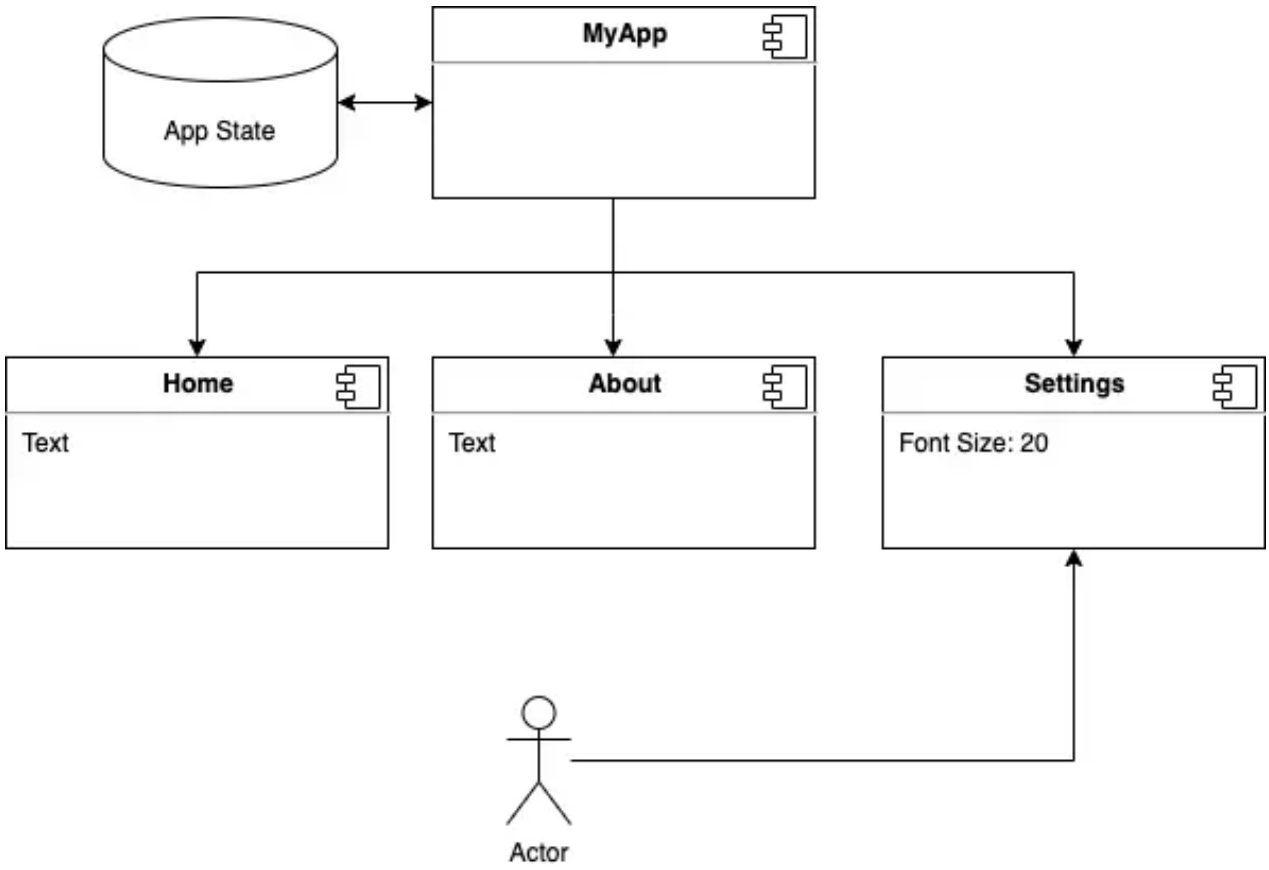
1. Home
2. About
3. Settings

Từ màn setting screen, bạn có thể đổi font chữ, các màn home , abount screen sẽ bị đổi font. Đầu tiên, bạn hãy cài thư viện





1. Khái niệm Global State Nếu bạn đã sử dụng widget statefull , bạn sẽ về state và cách dùng state trong fullter như thế nào . Nhưng chúng tôi muốn dùng global state để có thể dùng ở các màn hình hoặc có thể truy cập data. Bạn có thể nhìn sơ đồ trực quan sau :

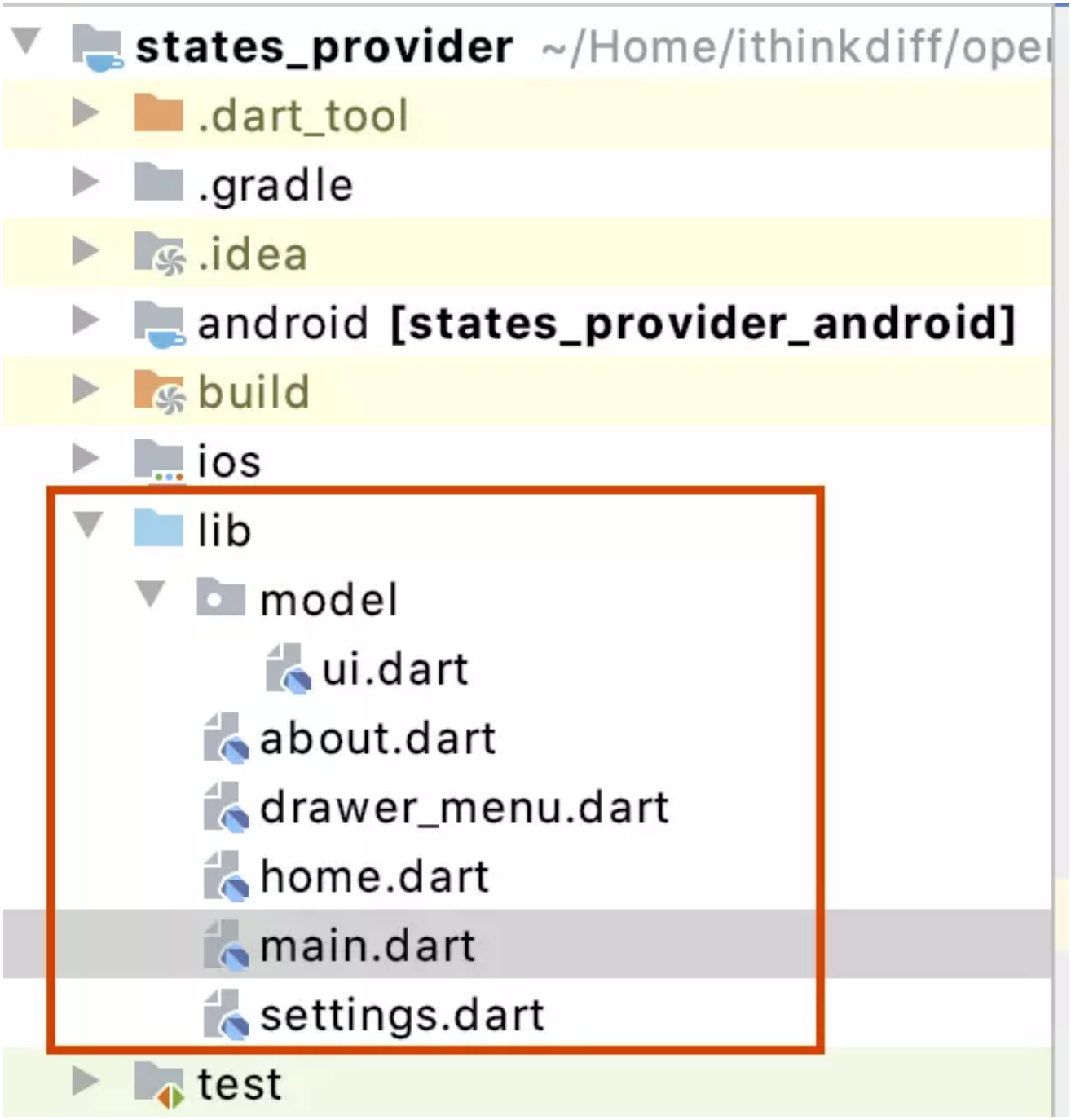


1. My App là main widget là nơi chúng ta có bind **App State**
2. Tất cả màn hình đều là con của widget MyApp.
3. Nhưng chúng ta có global state thì chúng ta có thể truy cập ở bất cứ nơi nào 1 cách dễ dàng .

2. Tạo App

Tôi sẽ dùng Android Studio để tạo app với tên "states_providers". Bạn sẽ tạo 5 file trong thư mục **lib**:

1. model/ui.dart
2. about.dart
3. drawer_menu.dart
4. home.dart
5. settings.dart



Sau đó bạn sẽ cài trong file **pubspec.yaml**:

flutter_lorem: ^1.1.0 provider: ^4.3.2+2

Chúng ta cần flutter_lorem để thực hiện random text.

3. Important concept

Để dùng được global state bằng Provider , chúng ta cần hiểu 3 class sau :

1. ChangeNotifier
2. ChangeNotifierProvider
3. Consumer

ChangeNotifier: Nó có nhiệm vụ thông báo cho người nghe.

ChangeNotifierProvider: Nó sẽ lắng nghe khi **ChangeNotifier.notifyListeners** được gọi và thông báo tới các hàm build liên quan .

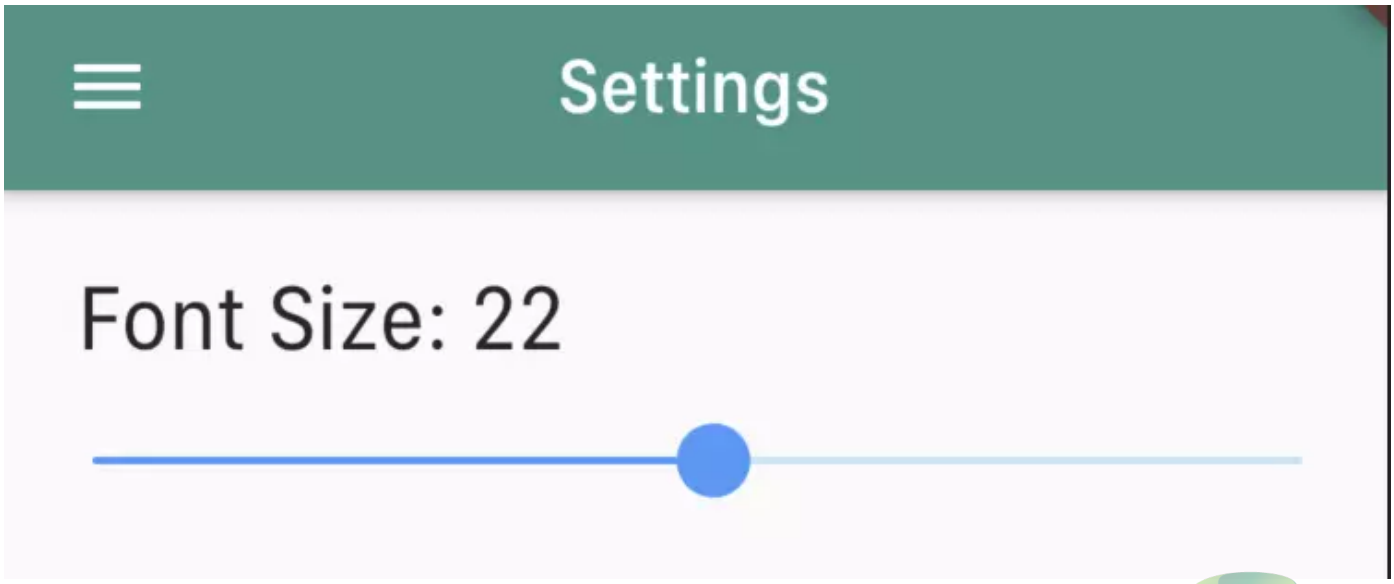
Consumer: đơn giản nó chỉ là một Widget do thư viện cung cấp . Chúng ta dùng widget này để lấy ra object thay vì phải gọi **Provider.of**. Bạn tham khảo về class này ở đây nhé : https://pub.dev/documentation/provider/latest/provider/Consumer-class.html .

4. Tạo Model :

Mở file model/ui.dart và viết đoạn code sau :

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

class UI with ChangeNotifier {
  double _fontSize = 0.5;

  set fontSize(newValue) {
    _fontSize = newValue;
    notifyListeners();
  }

  double get fontSize => _fontSize * 30;

  double get sliderFontSize => _fontSize;
}
```

Ở đây tôi tạo class UI implement class **ChangeNotifier**, tạo biến private font_size và một số method có thể truy cập hoặc thay đổi value.



Trong hàm **set fontSize(newValue)**, bạn sẽ thấy có hàm **notifyListeners()** ở cuối . Nếu value fontSize thay đổi , nó thông báo cho người nghe của nó . Nếu bạn ko viết hàm này thì sẽ không có điều gì xảy ra , nó rất là quan trọng. Một điều khác là slider value có phạm vi giá trị từ 0.0 -> 1.0 Nhưng tôi muốn điều chỉnh font với kích thước có thể đọc được . Tôi sẽ *30 giá trị của value nó lên :

```
double get fontSize => _fontSize * 30;
```

1. Thay đổi Main.dart

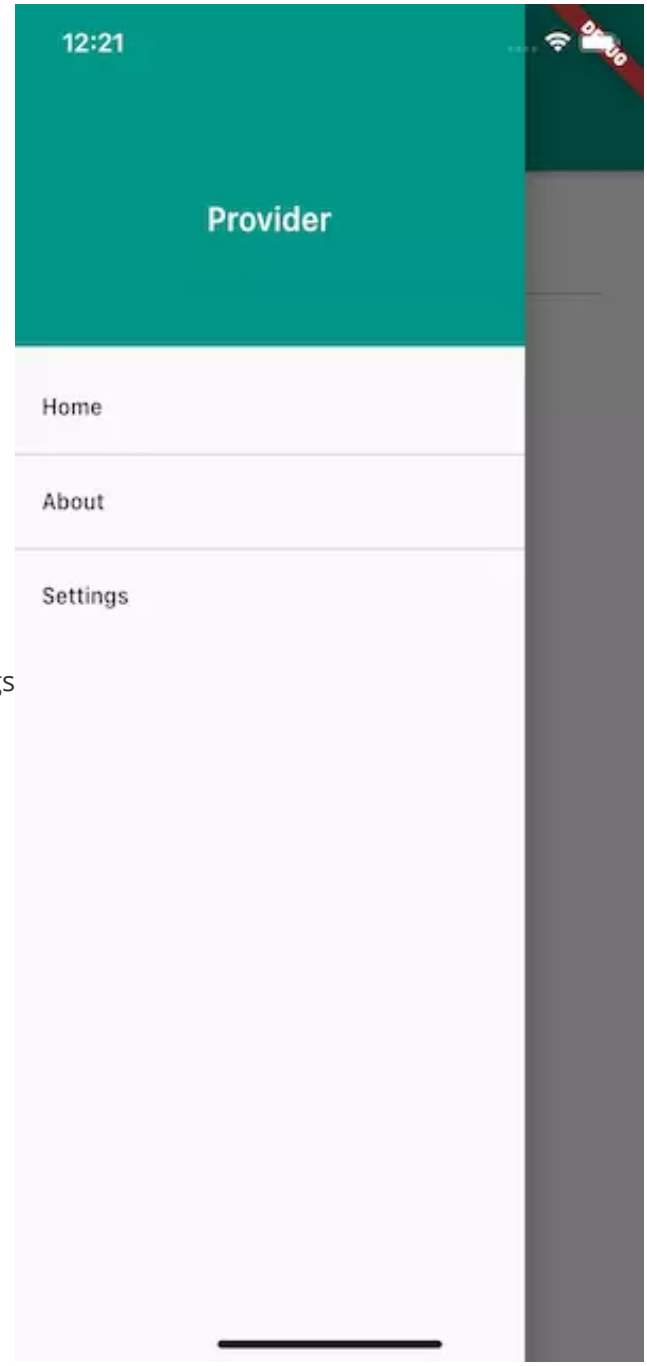
Mở file main.dart , xoá tất cả các đoạn code thay bằng đoạn code sau :

```
import 'package:flutter/material.dart';
import 'package:states_provider/home.dart';
import 'package:states_provider/about.dart';
import 'package:states_provider/settings.dart';
import 'package:provider/provider.dart';
import 'package:states_provider/model/ui.dart';

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (_) => UI()),
      ],
      child: MaterialApp(
        initialRoute: '/',
        routes: {
          '/': (context) => Home(),
          '/about': (context) => About(),
          '/settings': (context) => Settings(),
        },
      ),
    );
  }
}
```

Trong MyApp, chúng ta tạo widget **MultiProvider** , trong list **providers** chúng ta truyền `ChangeNotifierProvider(create: (_) => UI())`. UI là model class và dùng ChangeNotifierProvider để tạo instance của class UI. Trong phần child của widget MultiProvider tôi sẽ tạo widget MaterialApp để config routing. Trong app nếu bạn muốn có nhiều provider thì bạn thêm `ChangeNotifierProvider(create: (_) => YOUR_DATA_MODEL())` vào trong list **providers**.

5. Tạo drawer menu

Trong menu sẽ có 3 item : Home, About, Settings



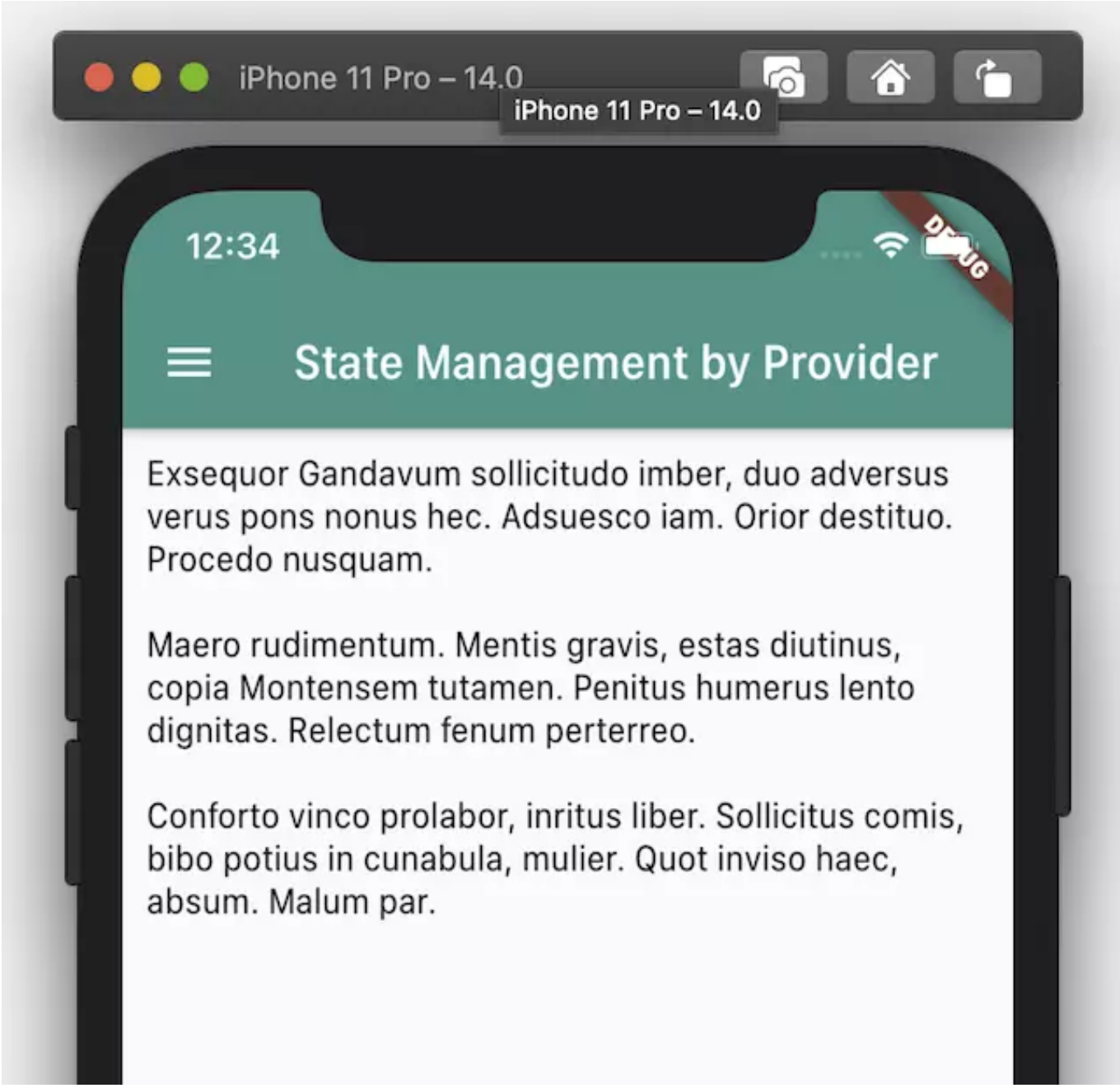
Mở file drawer_menu.dart và thêm đoạn code sau :

```
1  import 'package:flutter/material.dart';
2
3  const kTitle = 'Provider';
4
5  class DrawerMenu extends StatelessWidget {
6    @override
7    Widget build(BuildContext context) {
8      return Drawer(
9        child: ListView(
10         padding: EdgeInsets.zero,
11         children: <Widget>[
12           DrawerHeader(
13             child: Center(
```

```
              child: Text(
                kTitle,
                style: TextStyle(
                  fontSize: Theme.of(context).textTheme.title.fontSize,
                  color: Colors.white,
                ),
              ),
            ),
            decoration: BoxDecoration(
              color: Colors.teal,
            ),
          ),
          getListTile('Home', onTap: () {
            Navigator.pushReplacementNamed(context, '/');
          }),
          getLine(),
          getListTile('About', onTap: () {
            Navigator.pushReplacementNamed(context, '/about');
          }),
          getLine(),
          getListTile('Settings', onTap: () {
            Navigator.pushReplacementNamed(context, '/settings');
          }),
        ],
      ),
    );
  }

  Widget getLine() {
    return SizedBox(
      height: 0.5,
      child: Container(
        color: Colors.grey,
      ),
    );
  }

  Widget getListTile(title, {Function onTap}) {
    return ListTile(
      title: Text(title),
      onTap: onTap,
    );
  }
```

6. Tạo màn Home

Bây giờ , mình sẽ tạo màn home screen và dùng để show random text :



Mở file home.dart và thêm đoạn code sau :

```dart
import 'package:flutter/material.dart';
import 'package:states_provider/drawer_menu.dart';
import 'package:flutter_lorem/flutter_lorem.dart';
import 'package:provider/provider.dart';
import 'package:states_provider/model/ui.dart';

const kAppTitle = 'State Management by Provider';
const kStateType = 'Provider';

class Home extends StatelessWidget {
  String text = lorem(paragraphs: 3, words: 50);

  @override
```

```
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(kAppTitle),
        backgroundColor: Colors.teal,
      ),
      drawer: DrawerMenu(),
      body: Container(
        margin: EdgeInsets.all(10),
        child: Consumer<UI>(    (1)
          builder: (context, ui, child) {    (2)
            return RichText(
              text: TextSpan(
                text: text,
                style: TextStyle(fontSize: ui.fontSize, color: Colors.black),
              ),
            );
          },
        ),
      ),
    );
  }
}
```

Đoạn code trên rất dễ hiểu . Cần lưu ý rằng nhớ sử dụng (1) **Consumer** , đoạn này dùng để giúp ta lấy ra được model của class UI. (2) Ở đoạn code này chúng ta truyền 1 function với 3 tham số : **context**, **ui**, and **child**. Trong đó **ui** là một instance của class UI và đã được binded main widget **MyApp**.

Bây giờ trong builder function , tôi sẽ đặt widget :

```
TextSpan(
              text: text,
              style: TextStyle(fontSize: ui.fontSize, color: Colors.black),
)
```

để hiển thị kết quả front chữ .

7. Màn hình About

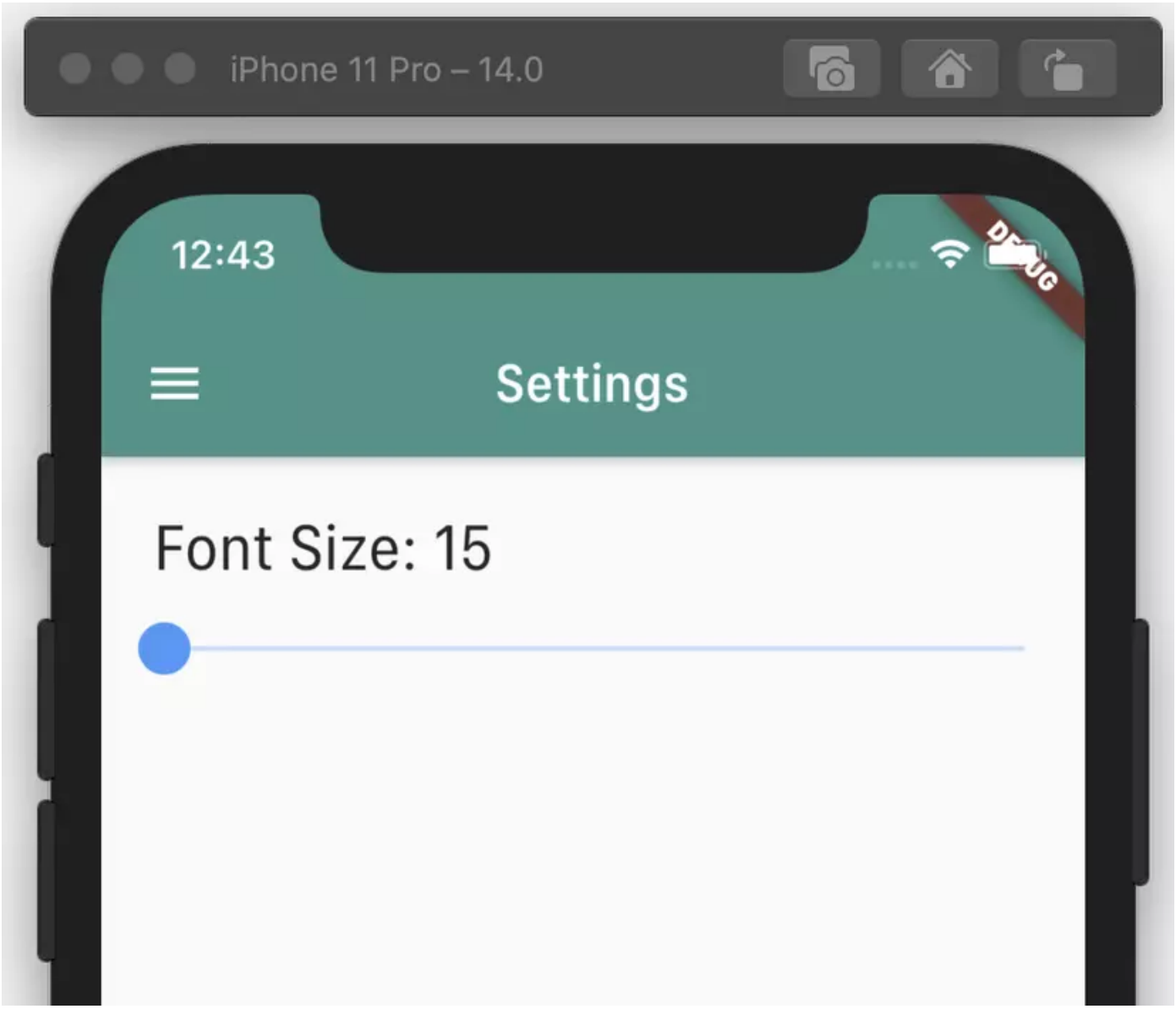
Tương tự như màn hình Home , bạn mở file about.dart và thêm đoạn code sau :

```
import 'package:flutter/material.dart';
import 'package:flutter_lorem/flutter_lorem.dart';
import 'package:states_provider/drawer_menu.dart';
import 'package:provider/provider.dart';
```

```dart
import 'package:states_provider/model/ui.dart';

class About extends StatelessWidget {
 String text = lorem(paragraphs: 3, words: 50);

 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       title: Text('About'),
       backgroundColor: Colors.teal,
     ),
     drawer: DrawerMenu(),
     body: Container(
       margin: EdgeInsets.all(10.0),
       child: Consumer<UI>(
         builder: (context, ui, child) {
           return RichText(
             text: TextSpan(
               text: text,
               style:
                   TextStyle(fontSize: ui.fontSize, color: Colors.lightBlue),
             ),
           );
         },
       ),
     ),
   );
 }
}
```

8. Màn Setting

Đây là màn hình mà người dùng sẽ thay đổi front chữ :



Mở file setting.dart và thêm đoạn code sau :

```dart
 import 'package:flutter/material.dart';
import 'package:states_provider/drawer_menu.dart';
import 'package:provider/provider.dart';
import 'package:states_provider/model/ui.dart';

class Settings extends StatelessWidget {
@override
Widget build(BuildContext context) {
 return Scaffold(
   appBar: AppBar(
     backgroundColor: Colors.teal,
     title: Text('Settings'),
   ),
   drawer: DrawerMenu(),
   body: Consumer<UI>(builder: (context, ui, child) {
     return Column(
```

```
        crossAxisAlignment: CrossAxisAlignment.start,
        children: <Widget>[
          Padding(
            padding: EdgeInsets.only(left: 20, top: 20),
            child: Text(
              'Font Size: ${ui.fontSize.toInt()}',
              style: TextStyle(
                  fontSize: Theme.of(context).textTheme.headline5.fontSize),
            ),
          ),
          Slider(
              min: 0.5,
              value: ui.sliderFontSize,
              onChanged: (newValue) {
                ui.fontSize = newValue;
              }),
        ],
      );
    }),
  );
}
}
```

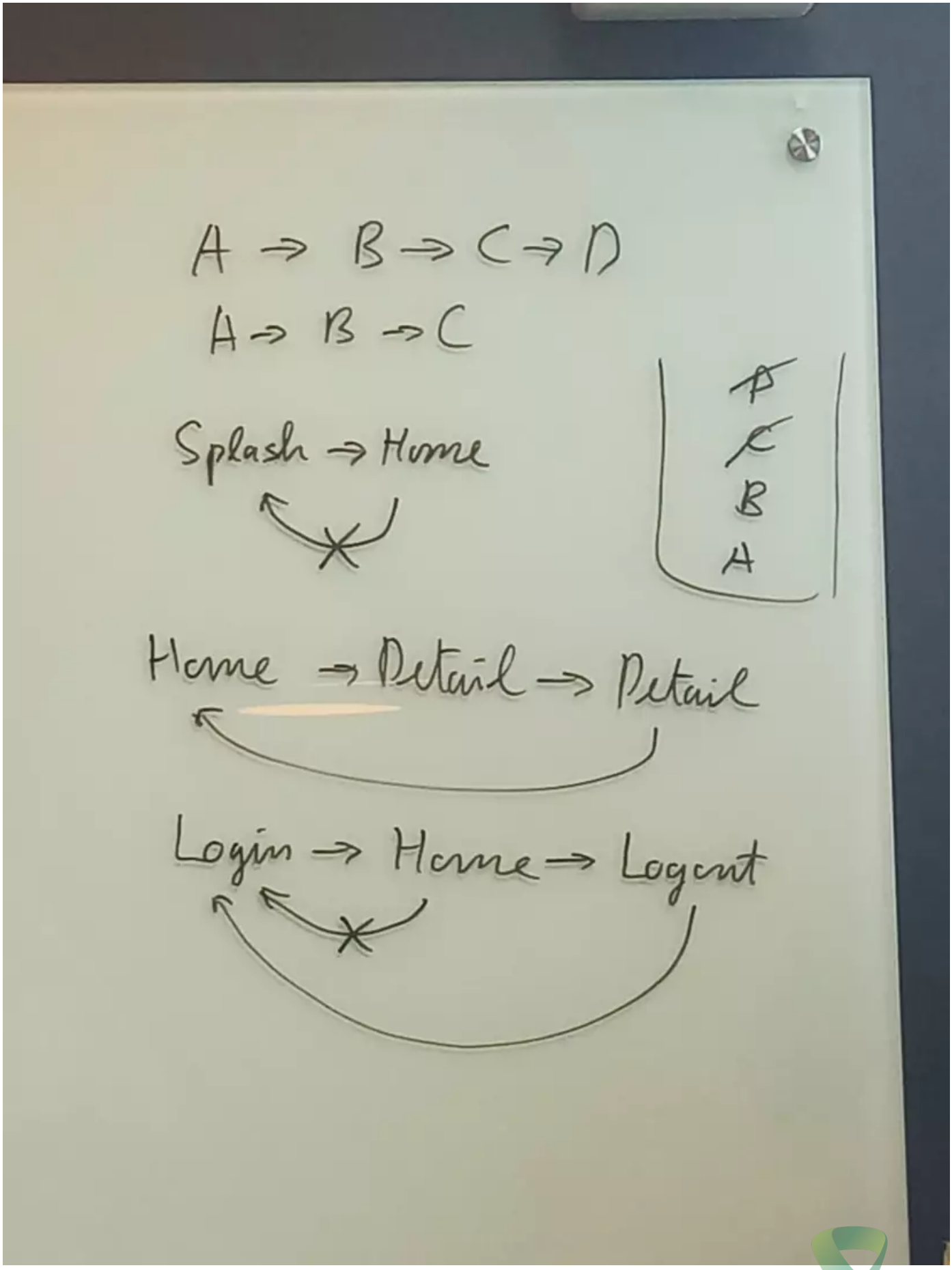
Màn này khác với màn home là màn này vừa có thể truy cập vừa có thể update data front size bằng đoạn code : `ui.fontSize = newValue.`. Khi mở các màn hình home, about front chữ sẽ được cập nhật . Đây là ví dụ đơn giản để các bạn hiểu cách dùng global state bằng thư viện provider. Bài viết của mình đến đây là kết thúc.

Tài liệu tham khảo

```
https://medium.com/level-up-programming/how-to-use-provider-in-flutter-f4998acb4702
```

Navigation trong Flutter

Flutter cung cấp widget Navigator để quản lý và thao tác với stack khi thực hiện điều hướng các màn hình.



Trong quá trình phát triển app mobile chúng ta sẽ có một số case điều hướng cơ bản cần phải xử lý như hình bên trên, hãy xem flutter hỗ trợ giải quyết các case điều hướng đó như thế nào nhé

Note nhỏ

Navigator cung cấp 2 loại function là

```
Navigator.pushNamed(context, string)

Navigator.of(context).pushNamed(string)
```

hai cách gọi bên trên là tương đương và nếu bạn đọc source thì `Navigator.pushNamed(context, string)` là hàm static gọi đến `Navigator.pushNamed(context, string)`

1. push, pop

Hai hàm cơ bản nhất và hay sử dụng nhất khi thực hiện các thao tác navigation

push

Thực hiện push widget vào stack của navigator, mỗi lần gọi hàm là một lần push widget vào stack

Gồm có 2 loại là:

- `push(context, route)`
- `pushNamed(context, string)`

push(context, route) aka direct navigation

```
Navigator.push(
    context, MaterialPageRoute(builder: (context) => Screen1()));

// or

Navigator.push(
    context, MaterialPageRoute(builder: (context) {
    // do something
        return Screen1();
    }));
```

Cách này cho bạn kiểm soát tốt hơn việc khở tạo màn hình mới, giúp bạn có thể thực hiện thêm thao tác tiền xử lý, hoặc truyền param cho màn mới, ...

pushNamed(context, string) with static navigation

```
class Routes {
  static final String screen1 = "/screen1";
  static final String screen2 = "/screen2";
}

MaterialApp(
    routes: {
        Routes.screen1: (context) => Screen1(),
        Routes.screen2: (context) => Screen2(),
    }
)

Navigator.pushNamed(context, Routes.screen1);
```

Bên trên là định nghĩa hết các name trong 1 class Routes, ngoài ra bạn có thể định nghĩa name trong cục bộ widget

```
class Screen1 extends StatelessWidget {
  static final String screen1 = "/screen1";
}
```

Cách này giúp bạn định nghĩa route ngắn gọn, nhưng bị giới hạn khi routeNamed sẽ trả về constructor cố định

pushNamed(context, string) with dynamic navigation

Cách bên trên giới hạn chúng ta ở việc linh động và ko thể custom construcotr của navigation thì sử dụng `onGenerateRoute` sẽ khắc phục các nhược điểm đó

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: Screen1(),
      ),
      onGenerateRoute: (settings) {
        switch (settings.name) {
          case Routes.screen2:
            return MaterialPageRoute(builder: (_) => Screen2());
            break;

```

```
          case Routes.screen3:
            return MaterialPageRoute(builder: (_) =>
                Screen3(
                    settings.arguments
                ));
            break;

          default:
            return MaterialPageRoute(builder: (_) => Screen1());
        }
      },
    );
  }
}
```

Các bạn có thể khai báo `initialRoute: name` thay vì khai báo `home: widget` trong `MaterialApp`

```
    return MaterialApp(
      initialRoute: Routes.screen1,
      onGenerateRoute: (settings) {
          ...
      },
```

pop(context)

Thực hiện pop widget ở trên cùng của stack navigator, mỗi lần gọi là một lần pop cho đến khi stack hết widget.

```
Navigator.pop(context);
```

2. Truyền data từ A push B

Từ màn A, mở màn B và bạn muốn truyền thêm một vài thông tin thì có 2 cách để thực hiện:

- Truyền qua constructor của B
- Truyền qua argumments

Truyền qua constructor

Để thực hiện cách này thì ở class A bạn sẽ cần phải dùng `push(context, route)`.

Ở bên class B thì chỉ cần gọi var là có giá tị

```
classs B {
    final String title;

    B({@require this.title});
}

class A {
    toB() {
        Navigator.push(
            context, MaterialPageRoute(builder: (context) => B('from A to B')));
    }
}
```

Truyền qua arguments

Các hàm push có hỗ trợ optional param arguments đều hỗ trợ việc truyền data.

Các bạn có thể dùng `push(context, route, arguments)` hoặc `pushNamed(context, string, arguments)` để thực hiên truyền từ A.

Tại B để nhận thì cần lấy ra từ arrguments.

```
class A {
    pushNamed(context, "/B", arguments: "from A to B");
}

class B {
    String args = ModalRoute.of(context).settings.arguments
}
```

Lưu ý: do arguments là một kiểu object nên khi muốn truyền nhiều loại data khác nhau thì cần phải tạo object wrap hết những type bạn cần truyền.

3. return data từ B về A

Để truyền dữ liệu từ B về A thì dùng `pop(context, result)` với param result là dữ liệu bạn muốn trả về.

Tại A, hàm push trả về future nên việc await hàm push sẽ nhận được dữ liệu từ B

```
class B {
    Navigator.pop(context, result);
}

class A {
    final result = await Navigator.push(B)
}
```

4. Các hàm push khác

Navigator còn có một số hàm push khác để cho những case cần custom flow navigation như sau:

- `pushAndRemoveUntuil / pushNamedAndRemoveUntil`
- `pushReplacement / pushReplacementNamed`
- `popAndPushNamed`

ở bên trên mình đã giải thích về `push/ pushNamed` nên dưới đây mình chỉ nói về ý nghĩa của các hàm này chứ không nói đến cách thức khác nhau nữa.

pushAndRemoveUntil / pushNamedAndRemoveUntil (context, route/string, bool)

Thực hiện thêm widget vào stack và pop các widget trong stack cũ cho đến khi `bool == true`

Về mặt UI sẽ nhìn thấy enter animation của push widget mới vào.

```
Navigator.pushAndRemoveUntil(
    context,
    MaterialPageRoute(builder: (BuildContext context) => Screen1()),
    ModalRoute.withName('/first'),
);
```

Nếu bạn muốn pop hết các widget sẵn có trong stack thì có thể return false ở param bool

Use case:

- Sau khi thực hiện các bước purchase, push màn status và pop hết các màn purchase
- Sau khi thực hiện các thao tác và nhấn logout, pop hết các màn và push login

pushReplacement / pushReplacementNamed

Thực hiện push widget vào stack và pop widget hiện tại của stack

Về mặt UI sẽ nhìn thấy **enter animation** của push widget mới vào.

Use case:

- Từ màn splash mở màn Home
- Từ màn Login, login thành công mở màn Home

popAndPushNamed

Thực hiện pop widget hiện tại của stack và push widget mới vào. Về ý nghĩa thì giống `pushReplacement`

Tuy nhiên về mặt UI sẽ nhìn thấy **exit animation** của widget hiện tại bị pop

Use case:

- Khi thực hiện xem item list, mở filter, chọn và apply filter thì pop màn filter và push màn item list

5. Các hàm pop khác

Navigator còn có một số hàm pop khác để cho những case cần custom flow navigation như sau:

- popUntil
- canPop
- maybePop

Chúng ta cùng đi vào từng loại nhé

popUntil(bool)

Hàm này dễ hiểu rồi, pop widget trong stack cho đến khi `bool == true`

canPop

return false nếu đây là widget đầu tiên trong navigator stack, hay stack size = 1. Nếu stack size > 1 thì return true.

maybePop = if(canPop) pop

Nếu stack size lớn hơn 1 thì mới thực hiện pop còn không thì thôi

6. Các hàm khác

Các hàm sau của Navigator đều cần param route (route = MaterialPagedRoute(builder:)). Nên để thực hiện thì bạn cần có refer đến route tương ứng mà muốn gọi hàm.

Hiện tại chưa thể get stack của navigator nên việc này sẽ hơi rắc rối một chút.

- `replaceRoute (context, oldRoute, newRoute)`
- `replaceRouteBelow (context, anchorRoute, newRoute)`
- `removeRoute (context, route)`
- `removeRouteBelow (context, anchorRoute)`

replaceRoute (context, oldRoute, newRoute)

replace oldROute trong stack bằng newRoute

replaceRouteBelow (context, anchorRoute, newRoute)

replace route ngay dưới anchorRoute trong stack bằng newRoute

removeRoute (context, route)

remove route trong stack

removeRouteBelow (context, anchorRoute)

remove route ngay dưới anchorRoute trong stack

7. ModalRoute

ModalRoute có nhiều hàm tiện ích các bạn có thể dọc thêm và sử dụng, ở đây mình sẽ chỉ giới thiệu một số ví dụ

get arguments

Như bên đã giới thiệu thì khi truyền arguments từ A sang B thì để get arguments ở B chúng ta cần dùng ModalRoute

```
final String args = ModalRoute.of(context).settings.arguments;
```

get name

Để get name của route hiện tại chúng ta sử dụng `ModalRoute`

```
final name = ModalRoute.of(context).settings.name;
```

so sánh route name

```
bool = ModalRoute.withName(string);
```

8. handle back button

Back button mặc định sẽ pop mà không phải lúc nào bạn cũng muốn như vậy nên việc custom lại hành vi khi click back button là rất thường gặp và trong Flutter chúng ta sẽ làm như sau

```
@override
Widget build(BuildContext context) {
  return WillPopScope(
    onWillPop: _onBackPressed, // function here
    child: Scaffold(
      body: Center(
        child: Text("Home"),
      ),
    ),
  );
}
```

Kết

Bài này mình đã giới thiệu tới các bạn về widget Navigator trong Flutter để xử lý các tác vụ navigation. Tùy theo yêu cầu cụ thể khi phát triển mà bạn sẽ chọn cho mình phương án phù hợp nhất.

** Tham khảo tài liệu tại Viblo.asia

Shared Preferences trong Flutter

Để lưu trữ các dữ liệu ở local trong ứng dụng Flutter, ngoài cách lưu bằng sqlite, chúng ta còn thể lưu dữ liệu vào Shared Preferences

1. Sơ lược về Shared Preferences trong Flutter

- Dùng để lưu những tập dữ liệu nhỏ dưới dạng key-value
- Các loại dữ liệu có thể lưu như là int, double, bool, String and List
- Các dữ liệu được lưu lại trong một file .xml và được lưu vào trong bộ nhớ đệm của máy
- Các dữ liệu chúng ta có thể dùng để lưu như là các thông số về Settings, token,, ...

2. Cách sử dụng

- Thêm thư viện vào trong file pubspect.yaml:

```
shared_preferences: any
```

Vì các hàm xử lý lưu dữ liệu trong shared_preferences đều là các hàm Future, nên chúng ta cần dùng await để gọi:

- Hàm lưu dữ liệu

```dart
// Obtain shared preferences.
final prefs = await SharedPreferences.getInstance();

// Save an integer value to 'counter' key.
await prefs.setInt('counter', 10);
// Save an boolean value to 'repeat' key.
await prefs.setBool('repeat', true);
// Save an double value to 'decimal' key.
await prefs.setDouble('decimal', 1.5);
// Save an String value to 'action' key.
await prefs.setString('action', 'Start');
// Save an list of strings to 'items' key.
await prefs.setStringList('items', <String>['Earth', 'Moon', 'Sun']);
```

- Hàm đọc dữ liệu

```dart
// Try reading data from the 'counter' key. If it doesn't exist, returns null.
final int? counter = prefs.getInt('counter');
// Try reading data from the 'repeat' key. If it doesn't exist, returns null.
final bool? repeat = prefs.getBool('repeat');
// Try reading data from the 'decimal' key. If it doesn't exist, returns null.
final double? decimal = prefs.getDouble('decimal');
// Try reading data from the 'action' key. If it doesn't exist, returns null.
final String? action = prefs.getString('action');
// Try reading data from the 'items' key. If it doesn't exist, returns null.
final List<String>? items = prefs.getStringList('items');
```

- Nếu chúng ta muốn xóa bỏ dữ liệu đã được lưu

```dart
// Remove data for the 'counter' key.
final success = await prefs.remove('counter');
```

Xây dựng ứng dụng I'm Rich

I AM SO RICH!!!!

I PURCHASED THE MOST EX-PENSIVE APP ON THE PLAY STORE


Vietcombank