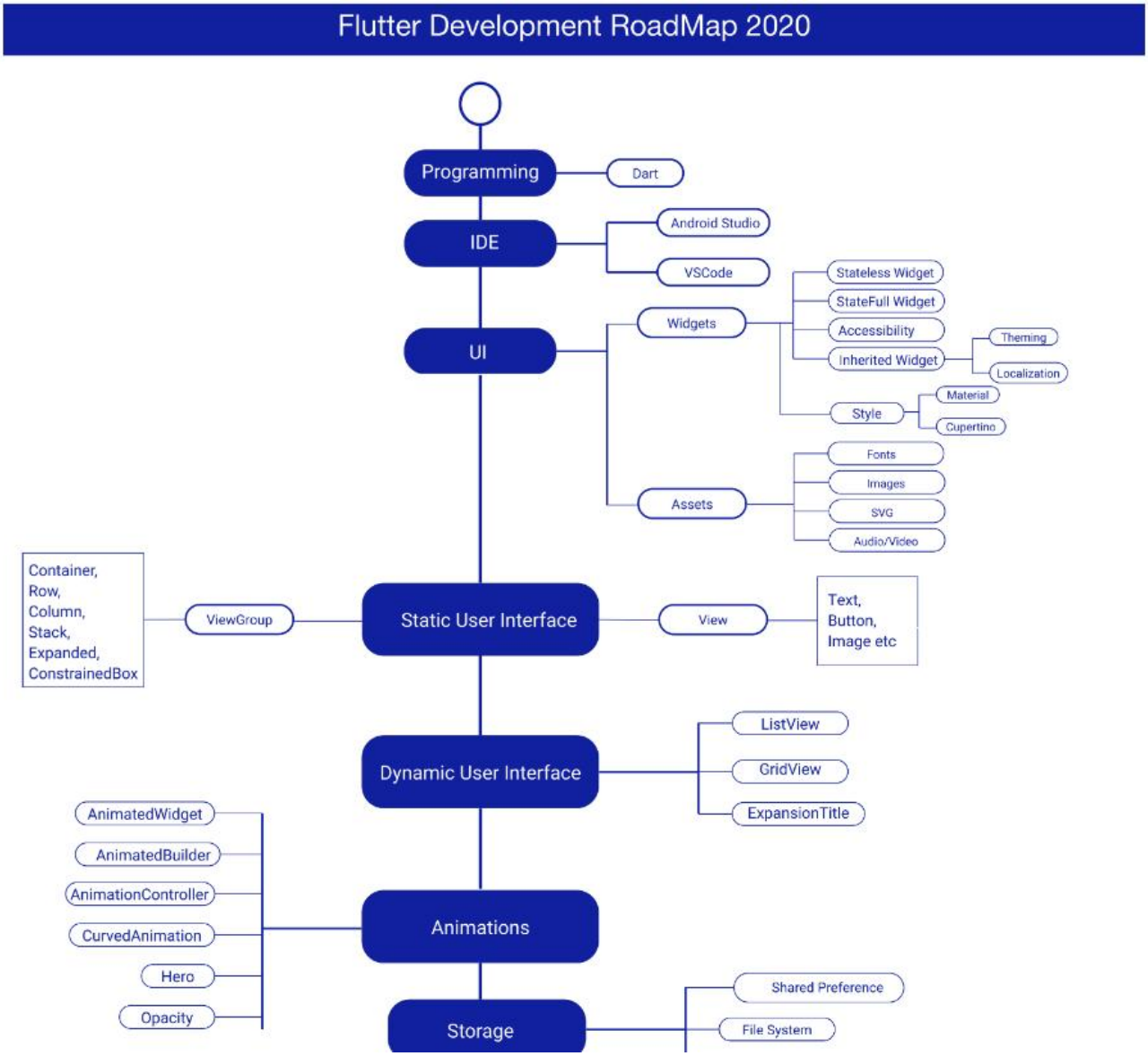
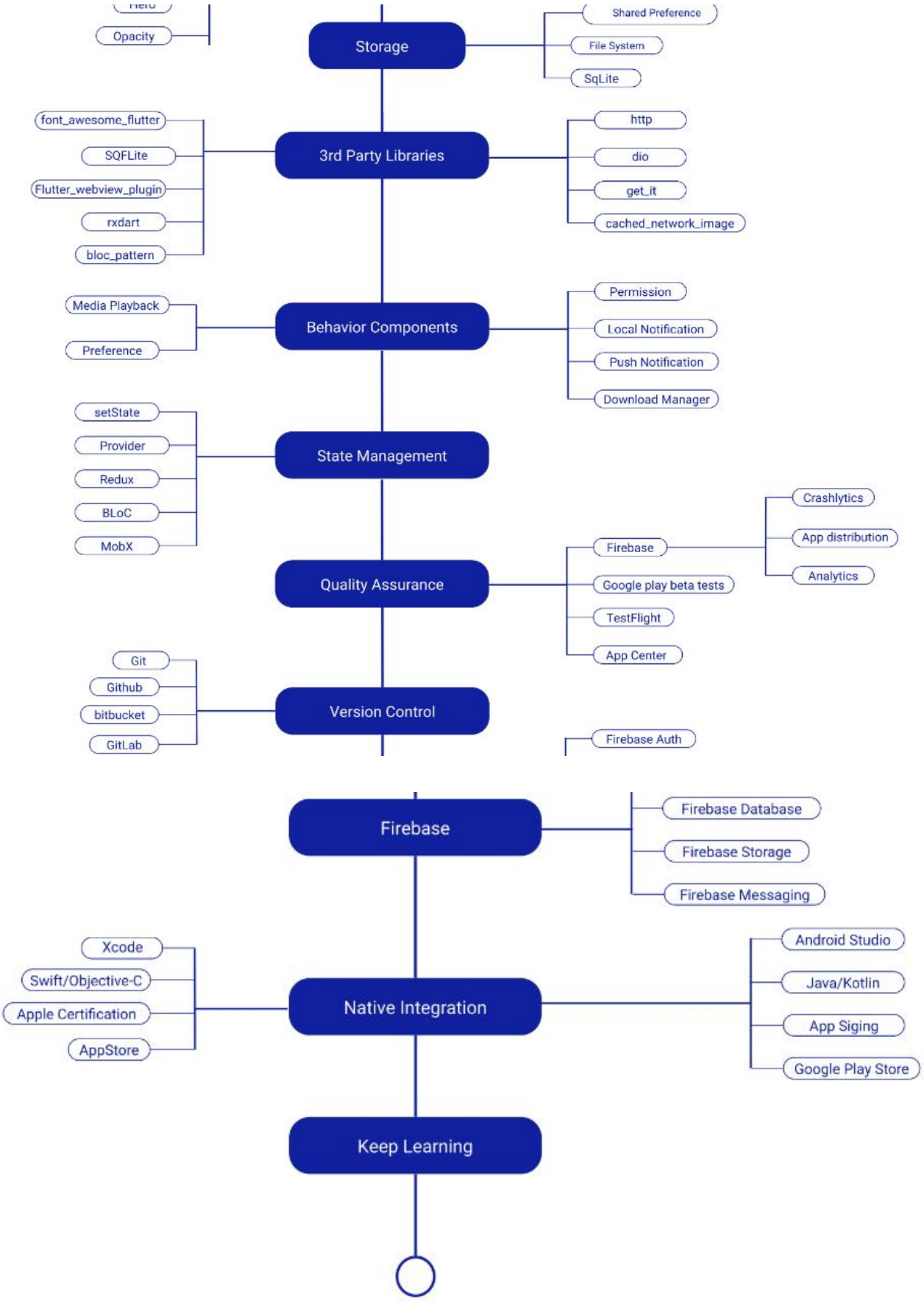
Dart 1.0 được phát hành bởi google vào ngày 14 tháng 11 năm 2013 và được viết bởi Lars Bak và Kasper Lund




Hero
Opacity
Storage
Shared Preference
File System
SqLite
font_awesome_flutter
SQFLite
Flutter_webview_plugin
rxdart
bloc_pattern
3rd Party Libraries
http
dio
get_it
cached_network_image
Media Playback
Preference
Behavior Components
Permission
Local Notification
Push Notification
Download Manager
setState
Provider
Redux
BLoC
MobX
State Management
Crashlytics
App distribution
Analytics
Firebase
Quality Assurance
Google play beta tests
TestFlight
App Center
Git
Github
bitbucket
GitLab
Version Control
Firebase Auth
Firebase
Firebase Database
Firebase Storage
Firebase Messaging
Xcode
Swift/Objective-C
Apple Certification
AppStore
Native Integration
Android Studio
Java/Kotlin
App Siging
Google Play Store
Keep Learning

CÁC WIDGET

1.Text

Đây là widget cơ bản giúp bạn format text trong ứng dụng của bạn, và thuộc tính quan trọng nhất của widget này là style

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

    ),
)
```

2.Column, Row

Đây là 2 widget giúp định hình 1 tập hợp các widget theo chiều dọc (vertically) hoặc chiều ngang (horizontally). 2 thuộc tính quan trọng nhất của 2 widget này là mainAxisAlignment và crossAxisAlignment.

Khi bạn sử dụng một Row, các widget con của nó được sắp xếp thành một hàng, theo chiều ngang. Vì vậy, trục chính của Row là nằm ngang. Sử dụng mainAxisAlignment trong Row cho phép bạn căn chỉnh các widget con của hàng theo chiều ngang (ví dụ: left, right). Trục chéo đến trục chính của Row là trục dọc. Vì vậy, sử dụng crossAxisAlignment trong Row cho phép bạn xác định cách con của nó được căn chỉnh theo chiều dọc. Trong một Cột thì ngược lại. Con của một Column được sắp xếp theo chiều dọc, từ trên xuống dưới (theo mặc định). Vì vậy trục chính của nó là thẳng đứng. Điều này có nghĩa là, việc sử dụng mainAxisAlignment trong một Column sẽ căn chỉnh các widget con của nó theo chiều dọc (ví dụ: top, bottom) và crossAxisAlignment xác định cách các con được căn chỉnh theo chiều ngang trong Column đó.

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

3. Stack

Cũng giống như Column hay Row, Stack cũng là 1 tập hợp các widget, tuy nhiên điểm khác biệt là các widget con trong stack có thể chồng lên nhau. Thường đi kèm với widget này là widget Positioned để căn chỉnh vị trí của từng widget trong Stack, hãy theo dõi ví dụ dưới để hiểu hơn về stack:

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

4. Container

Đây là 1 widget được mình thường xuyên được sử dụng vì tính hữu dụng của nó. Nó có các thuộc tính width, height giúp bạn có thể dễ dàng layout những view có kích thước đã xác định và đặc biệt là thuộc tính decoration giúp bạn trang trí lộng lẫy hơn cho view của bạn:

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

```
Container({
  Key key,
  this.alignment,
  this.padding,
  this.color,
  this.decoration,
  this.foregroundDecoration,
  double width,
  double height,
  BoxConstraints constraints,
  this.margin,
  this.transform,
  this.child,
  this.clipBehavior = Clip.none,
})
```

5. SizedBox

Đây là widget mình hay sử dụng để tạo khoảng cách giữa các widget hơn là việc layout một view:

```
Column(
  mainAxisAlignment: MainAxisAlignment.start,
  children: [
    SizedBox(
      height: 20,
    ),
    Text(
      'Hello World!',
      style: TextStyle(
        fontSize: 30,
        fontWeight: FontWeight.bold,
        color: Colors.black,
      ),
    ),
    SizedBox(
      height: 20,
    ),
    Text(
      'This is sample app',
      style: TextStyle(
        fontSize: 20,
        color: Colors.grey,
      ),
    )
  ],
),
```

6. SingleChildScrollView

Khi bạn có 1 Column hoặc 1 Row có height hoặc width vượt qua độ lớn màn hình điện thoại của bạn thì SingleChildScrollView widget xuất hiện giúp bạn giải quyết vấn đề này. Để widget này hoạt động đúng thì bạn cần phải set giá trị cho thuộc tính scrollDirection của nó (Axis.vertical là theo chiều dọc, Axis.horizontal theo chiều ngang):

```
        SingleChildScrollView(
          scrollDirection: Axis.horizontal,
                  child: Column(
            children: [
              LogoBox(),
              LogoBox(),
              LogoBox(),
              LogoBox(),
              LogoBox(),
              LogoBox(),
            ],
          ),
        ),
```

```
class LogoBox extends StatelessWidget {
 const LogoBox({
   Key key,
 }) : super(key: key);

 @override
 Widget build(BuildContext context) {
   return Container(
     width: 200,
     height: 200,
     margin: EdgeInsets.all(20),
     decoration: BoxDecoration(
       color: Colors.amber,
       border: Border.all(width: 1, color: Colors.black26),
       borderRadius: BorderRadius.circular(10),
     ),
     padding: EdgeInsets.all(50),
     child: FlutterLogo(
       size: 50,
     ),
   );
 }
}
```

7. Expanded

Đây là widget hay được xử dụng để chia bố cục của widget cha thành các phần với tỉ lệ tương ứng:

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

Widget trong Futter là gì?

Các widget là hệ thống phân cấp Class trong Framework Flutter. Widget là một thành phần giao diện người dùng. Các widget có thể là các phần tử, hoặc tập hợp nhiều phần tử sử dụng để xây dựng giao diện có thể được tái sử dụng.

State trong Flutter

State là thông tin có thể được đọc đồng bộ khi tiện ích con được xây dựng và có thể thay đổi trong suốt thời gian tồn tại của tiện ích con.

Nói cách khác, state của widget là dữ liệu của các đối tượng mà các thuộc tính (tham số) của nó đang duy trì tại thời điểm tạo ra nó (khi widget được vẽ trên màn hình). State cũng có thể thay đổi khi nó được sử dụng, chẳng hạn như khi một tiện ích CheckBox được nhấp vào, một dấu kiểm xuất hiện trên hộp.

Stateless Widget:

Các widget có state không thể thay đổi sau khi chúng được xây dựng được gọi là **Stateless Widget**. Các widget này là bất biến sau khi chúng được xây dựng, tức là bất kỳ thay đổi nào trong các biến, icon, Button hoặc dữ liệu truy xuất đều không thể thay đổi state của ứng dụng. Dưới đây là cấu trúc cơ bản của một **Stateless Widget**. **Stateless Widget** ghi đè phương thức build () và trả về một widget. Ví dụ: chúng tôi sử dụng text hoặc icon là ứng dụng trong đó state của tiện ích không thay đổi trong thời gian chạy. Nó được sử dụng khi giao diện người dùng phụ thuộc vào thông tin bên trong chính đối tượng. Các ví dụ khác có thể là Text, RaisedButton, Icon Buttons.

Vì vậy, chúng ta hãy xem đoạn mã nhỏ này cho chúng ta biết những gì. Tên của **Stateless Widget** là MyApp đang được gọi từ runApp () và kế thừa từ class StatelessWidget. Bên trong MyApp này, một hàm constructor được ghi đè và lấy BuildContext làm tham số. BuildContext này là duy nhất cho mỗi widget vì nó được sử dụng để định vị widget bên trong cây widget.

Stateful Widgets:

Các **Stateful Widgets** có thể thay đổi khi chúng được xây dựng được gọi là các widget state. Những state này có thể thay đổi được và có thể thay đổi nhiều lần trong thời gian tồn tại của chúng. Điều này đơn giản có nghĩa là state của một ứng dụng có thể thay đổi nhiều lần với các bộ biến, input, dữ liệu khác nhau. Dưới đây là cấu trúc cơ bản của một widget state. Class Stateful ghi đè createState () và trả về một state. Nó được sử dụng khi giao diện người dùng có thể thay đổi động. Một số ví dụ có thể là CheckBox, RadioButton, form, text.

Các lớp kế thừa "Stateful Widget" là bất biến. Nhưng State có thể thay đổi, thay đổi trong thời gian chạy khi người dùng tương tác với nó.

Vì vậy, hãy để chúng tôi xem những gì chúng tôi có trong đoạn mã này. Tên của Stateful Widget là MyApp được gọi từ runApp () và kế thừa một widget state. Trong lớp MyApp, chúng tôi ghi đè chức năng tạo state. Hàm createState () này được sử dụng để tạo state có thể thay đổi cho tiện ích con này tại một vị trí nhất định trong cây.

Platform specific widgets

- Flutter có những widget đặc biệt cụ thể cho từng nền tảng riêng biệt như là Android hoặc iOS.
- Các widget sử dụng cho Android được thiết kế dựa vào Material design guideline của Android OS. Các widget này được gọi là Material widgets.
- Các widget cho hệ điều hành iOS được xây dựng theo Human Interface guidelines bởi Apple có tên là Cupertino widgets.
- Các material widget được dùng phổ biến nhất được liệt kê theo danh sách dưới đây:

Scaffold, AppBar, BottomNavigationBar, TabBar, TabBarView, ListTile, RaisedButton, FloatingActionButton, FlatButton, IconButton, DropdownButton, PopupMenuButton, ButtonBar, TextField, Checkbox, Radio, Switch, Slider, Date & Time Pickers, SimpleDialog, AlertDialog.

- Các Cupertino widget được dùng phổ biến nhất được liệt kê theo danh sách dưới đây:

CupertinoButton, CupertinoPicker, CupertinoDatePicker, CupertinoTimerPicker, CupertinoNavigationBar, CupertinoTabBar, CupertinoTabScaffold, CupertinoTabView, CupertinoTextField, CupertinoDialog, CupertinoDialogAction, CupertinoFullscreenDialogTransition, CupertinoPageScaffold, CupertinoPageTransition, CupertinoActionSheet, CupertinoActivityIndicator, CupertinoAlertDialog, CupertinoPopupSurface, CupertinoSlider

Layout widgets

- Trong Flutter, một widget có thể được tạo bằng cách kết hợp giữa một hoặc nhiều widget. Để kết hợp nhiều widget vào một widget, Flutter cung cấp một lượng lớn các widget có chức năng layout. Ví dụ như một widget con có thể được canh lề bằng widget Center.
- Các layout widget phổ biến:
- Container – Là một cái hộp hình chữ nhật được trang trí bằng cách sử dụng widget BoxDecoration ở nền, đường biên và đổ bóng.
- Center – canh giữa widget con của nó.
- Row – xếp các con của nó theo chiều ngang.
- Column − xếp các con của nó theo chiều dọc.
- Stack – xếp các con của nó trồng lên nhau.

State maintenance widgets

- Trong Flutter, tất cả các widget được chia ra từ 2 widget chính đó là StatelessWidget hoặc StatefulWidget.
- Widget StatelessWidget không thông tin trạng thái (state) nào nhưng nó có thể chứa widget có StatefulWidget. Bản chất linh hoạt của ứng dụng là thông qua các hành vi tượng tác của các widget và cách trạng thái thay đổi trong tương tác. Ví dụ như, khi ta chạm một nút đếm thì nó sẽ tăng hoặc giảm trạng thái bên trong của nút đếm bằng một và widget sẽ tự động re-render lại widget sử dụng một thông tin trạng thái hoàn toàn mới.

Platform independent / basic widgets

Flutter cung cấp một lượng lớn các widget cơ bản để tạo một giao diện cơ bản cũng như phức tạp trong các nền tảng độc lập.

TEXTFIELD

1. Loại Keyboard

TextField cho phép bạn tùy chỉnh loại bàn phím hiển thị khi TextField được đưa vào tiêu điểm. Chúng tôi thay đổi thuộc tính keyboardType cho điều này.

```
TextField(
  keyboardType: TextInputType.number,
),
```

Các loại là:

1. **TextInputType.text** (Normal complete keyboard)
2. **TextInputType.number** (A numerical keyboard)
3. **TextInputType.emailAddress** (Normal keyboard with an “@”)
4. **TextInputType.datetime** (Numerical keyboard with a “/” and “:”)
5. **TextInputType.numberWithOptions** (Numerical keyboard with options to enabled signed and decimal mode)
6. **TextInputType.multiline** (Optimises for multi-line information)

```
TextField(
  decoration: InputDecoration(
      contentPadding:
          const EdgeInsets.all(10), //kich thuoc cua textfield
      filled: true, //background của textfield
      fillColor: const Color.fromARGB(
          255, 168, 193, 213), //background của textfield // Color.fromARGB
      hoverColor: Colors.red,
      border: OutlineInputBorder(
          borderRadius: BorderRadius.circular(20)), // OutlineInputBorder
      labelText: 'lable',
      hintText: 'hint',
      enabled: true,
      helperText: 'help',
      counterText: 'counter',
      icon: const Icon(Icons.first_page), //icon ngoai
      prefixIcon: const Icon(Icons.security_outlined), //icon dau
      suffixIcon: const Icon(Icons.thermostat), //icon cuoi
      focusedBorder: OutlineInputBorder(
          borderRadius: BorderRadius.circular(20),
          borderSide:
              const BorderSide(color: Colors.purple, width: 2.0))), // OutlineInputBorder // InputDecoration
  obscureText: true,
  keyboardType: TextInputType.number,
  controller: txtPass,
) // TextField
```

nút nhấn

```
ElevatedButton(
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => MenuScreen(),
      ), // MaterialPageRoute
    );
    if (txtPhone.text.isEmpty || txtPass.text.isEmpty) {
      showDialog(
          context: context,
          builder: ((context) {
            return AlertDialog(
              title: const Text('Nhập dữ liệu đê'),
              actions: [
                OutlinedButton(
                    onPressed: () {
                      Navigator.pop(context);
                    },
                    child: const Text('OK')), // OutlinedButton
              ],
            ); // AlertDialog
          }));
    } else {
      if (txtPhone.text.length < 10) {
        showDialog(
            context: context,
            builder: ((context) {
              return AlertDialog(
                title: const Text('Vui lòng nhập 10 kí tự'),
                actions: [
                  OutlinedButton(
```

```
                                OutlinedButton(
                                    onPressed: () {
                                      Navigator.pop(context);
                                    },
                                    child: const Text('OK')), // OutlinedButton
                                ],
                              ); // AlertDialog
                            }));
                      }
                    }
                  },
                  style: ButtonStyle(
                      backgroundColor: MaterialStateProperty.all(
                          const Color.fromARGB(255, 60, 68, 215))), // ButtonStyle
                  child: const Text(
                    'Đăng Nhập',
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                      fontSize: 18,
                    ), // TextStyle
                  ), // Text
                ) // ElevatedButton
```

```dart

import 'package:flutter/material.dart';
import 'package:flutter/widgets.dart';
import 'package:flutter_application_animation/menu_screen.dart';

class SignUpScreen extends StatefulWidget {
  @override
  State<StatefulWidget> createState() {
    return SignUpScreenState();
  }
}

class SignUpScreenState extends State<SignUpScreen> {
  TextEditingController txtPhone = TextEditingController();
  TextEditingController txtPass = TextEditingController();
  bool frag = true;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Padding(
        padding: const EdgeInsets.symmetric(horizontal: 24),
        child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            crossAxisAlignment: CrossAxisAlignment.stretch,
            children: [
```

JSON

Provider

```dart
import 'dart:convert';

import 'package:flutter/services.dart';
import 'package:flutter_application_animation/product_object.dart';

class ProductProvider{
  static Future<List<dynamic>> readJsonData() async {
    var jsonText = await rootBundle.loadString('data/ontap.json');
    var data = json.decode(jsonText);
    return data;
  }
  static Future<List<ProductObject>> getAllProduct() async {
    List<ProductObject> lsResult = [];
    List<dynamic> data = await readJsonData();
    lsResult = data.map((e) => ProductObject.fromJson(e)).toList();
    return lsResult;
  }
}
```

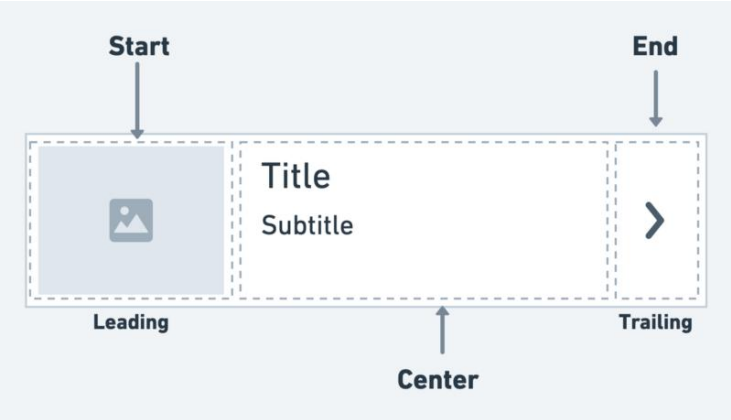
Object

```dart
child: Image.network(this.widget.product.picture),
this.widget.product.name,
'${this.widget.product.price} VND',
```

```dart
class ProductObject {
  final int id;
  final String name;
  final int price;
  final String description;
  final String picture;
  ProductObject({
   required this.id,
   required this.name,
   required this.price,
   required this.description,
   required this.picture});

  ProductObject.fromJson(Map<String, dynamic> res)
      : id = res['id'],
        name = res['name'],
        price = res['price'],
        description = res['description'],
        picture = res['picture'];
  Map<String, Object?> toJson() {
    return {
      'id': id,
      'name': name,
      'price': price,
      'description': description,
      'picture': picture,
    };
  }
}
```

```dart
class MenuScreenState extends State<MenuScreen> {
  List<ProductObject> lsProduct = [];
  final player = AudioPlayer();
  void _loadDS() async {
    final data = await ProductProvider.getAllProduct();
    setState(() {});
    lsProduct = data;
  }

  @override
  void initState() {
    super.initState();
    _loadDS();
  }
class ProductDetail extends StatefulWidget {
  ProductObject product;
  ProductDetail({Key? key, required this.product}) : super(key: key);

  @override
  State<StatefulWidget> createState() {
    return ProductDetailState();
  }
}
```

LISTVIEW



```dart
ListView.builder(
  padding: EdgeInsets.all(5),
  itemCount: 10 //lsProduct.length,
  itemBuilder: (context, index) => Card(
    child: ListTile(
      title: Text(
        'ten',//lsProduct[index].name,
        style: TextStyle(fontWeight: FontWeight.bold),
      ), // Text
      leading: //Image.network(lsProduct[index].picture),
      subtitle: Text(
        '',//'${lsProduct[index].price}',
        style: TextStyle(
          fontWeight: FontWeight.bold,
        ),
      ),
      onTap: () {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) =>
                ProductDetail(product: lsProduct[index]),
          ),
        );
      },
    ), // ListTile
  ), // Card
) // ListView.builder
```


RN_LIKE_APP
.dart_tool
.idea
android
app
gradle
.gitignore
build.gradle
gradle.properties
gradlew
gradlew.bat
local.properties
rn_like_app_android.iml
settings.gradle
build
ios
Flutter
Runner
Runner.xcodeproj
Runner.xcworkspace
.gitignore
lib
main.dart
test
web
.gitignore
.metadata
.packages
pubspec.lock
pubspec.yaml
README.md
rn_like_app.iml