



Flutter

Elyes Said

2020/2021



Présentation

ELYES SAID

Scrum Master ,

Développeur full Stack js, Flutter

elyes.said@viacesi.fr

Objectifs de la formation

Flutter

- ▶ Widgets
- ▶ Navigation
- ▶ Formulaires
- ▶ State Management
- ▶ Réseau (API)
- ▶ Stockage
- ▶ Génération des livrables
- ▶ ...

Dart

- ▶ Structures de contrôle
- ▶ Paramètres
- ▶ Types
- ▶ POO
- ▶ Tâches asynchrones
- ▶ Tests
- ▶ Outils
- ▶ ...

1

Technologies existantes



Applications natives

Android

- ▶ Java
- ▶ Kotlin (2017)

iOS

- ▶ Objective-C
- ▶ Swift (2014)

Avantages

- ▶ Performances
- ▶ Interface spécifique à chaque plateforme

Inconvénients

- ▶ 1 application par plateforme
- ▶ Coûts et temps de développement

Applications hybrides

- ▶ Cordova, Ionic, ...
- ▶ Basées sur des technologies web (HTML, JS, CSS, ...)
- ▶ Utilisent les WebView (navigateur web intégré)

Avantages

- ▶ Code partagé / réutilisation
- ▶ (Interface utilisateur spécifique à chaque plateforme)

Inconvénients

- ▶ Performances
- ▶ Sécurité

Applications cross-plateformes

- ▶ Exemples de framework : React Native, Xamarin, Flutter
- ▶ Objectif : générer 1application Android et iOS à partir du même code

Avantages

- ▶ Code partagé / réutilisation
- ▶ Compétences similaires au web

Inconvénients

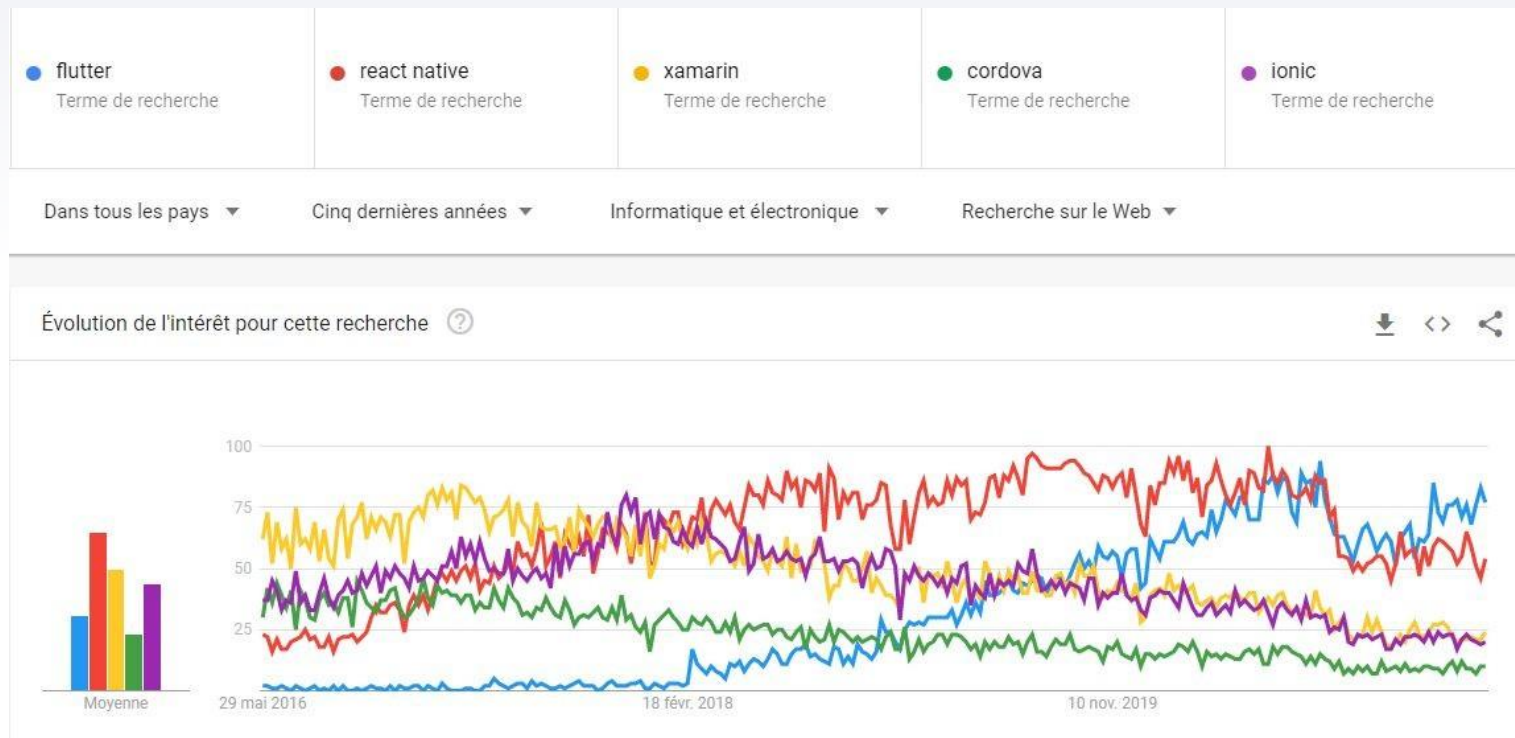
- ▶ Peut nécessiter d'écrire du code spécifique
- ▶ (Performances)

2

Quelques chiffres

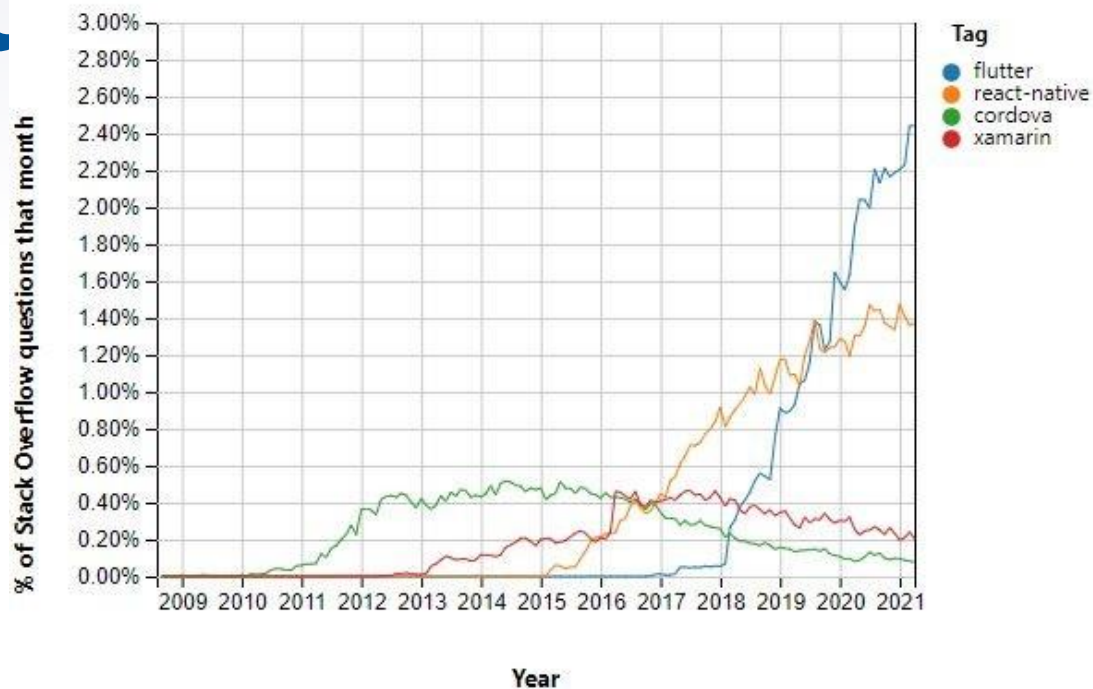


Google Trends



Source : <https://trends.google.fr/trends/explore?cat=5&date=today%205-y&q=flutter,react%20native,xamarin,cordova,ionic>

Stack Overflow Trends



Source : <https://insights.stackoverflow.com/trends?tags=flutter%2Creact-native%2Ccordova%2Cxamarin>

3

Flutter



Flutter

- ▶ **Framework UI** permettant de créer des applications :
 - ▶ **mobiles** : Android / iOS / Google Fuchsia
 - ▶ **web**
 - ▶ **desktop** : Windows / Mac / Linux
- ▶ Écrit en **langage Dart**
- ▶ Moteur de Flutter écrit principalement en C++
- ▶ Utilise la bibliothèque graphique Skia de Google
- ▶ Framework développé par Google, **1ère release décembre 2018**

Flutter Create

Flutter Create is a contest that challenges you to build something interesting, inspiring, and beautiful with Flutter using 5KB or less of Dart code



<https://flutter.dev/clock>

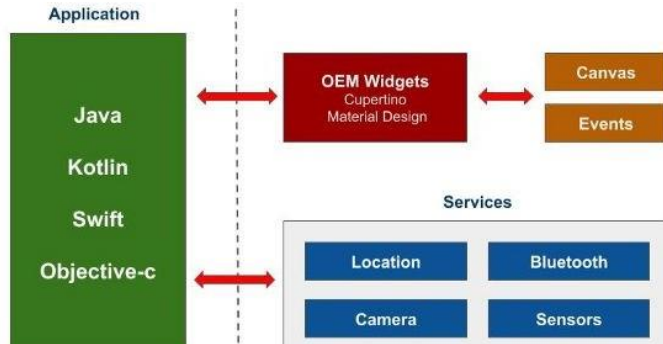
Flutter Clock Challenge

Build a beautiful clock face UI with Flutter for the Lenovo Smart Clock for a chance to win an iMac Pro, Lenovo Smart Display, or Lenovo Smart Clock.

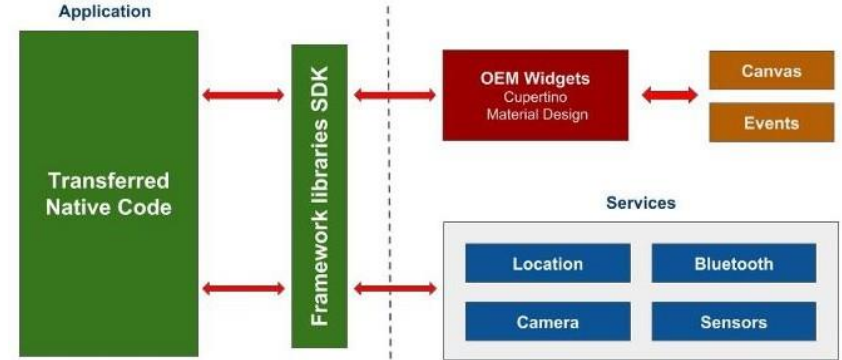


<https://flutter.dev/create>

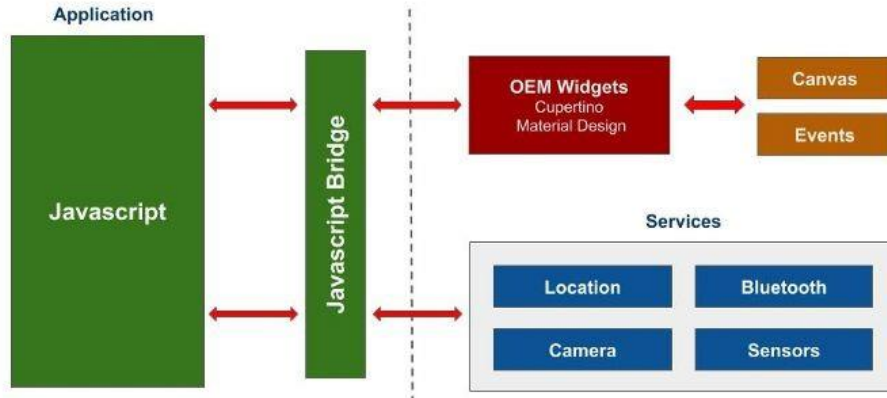
OEM SDKs



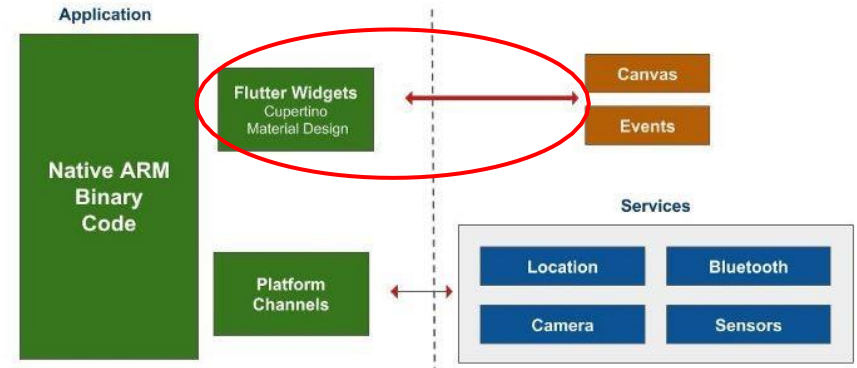
Cross-Platform Approach



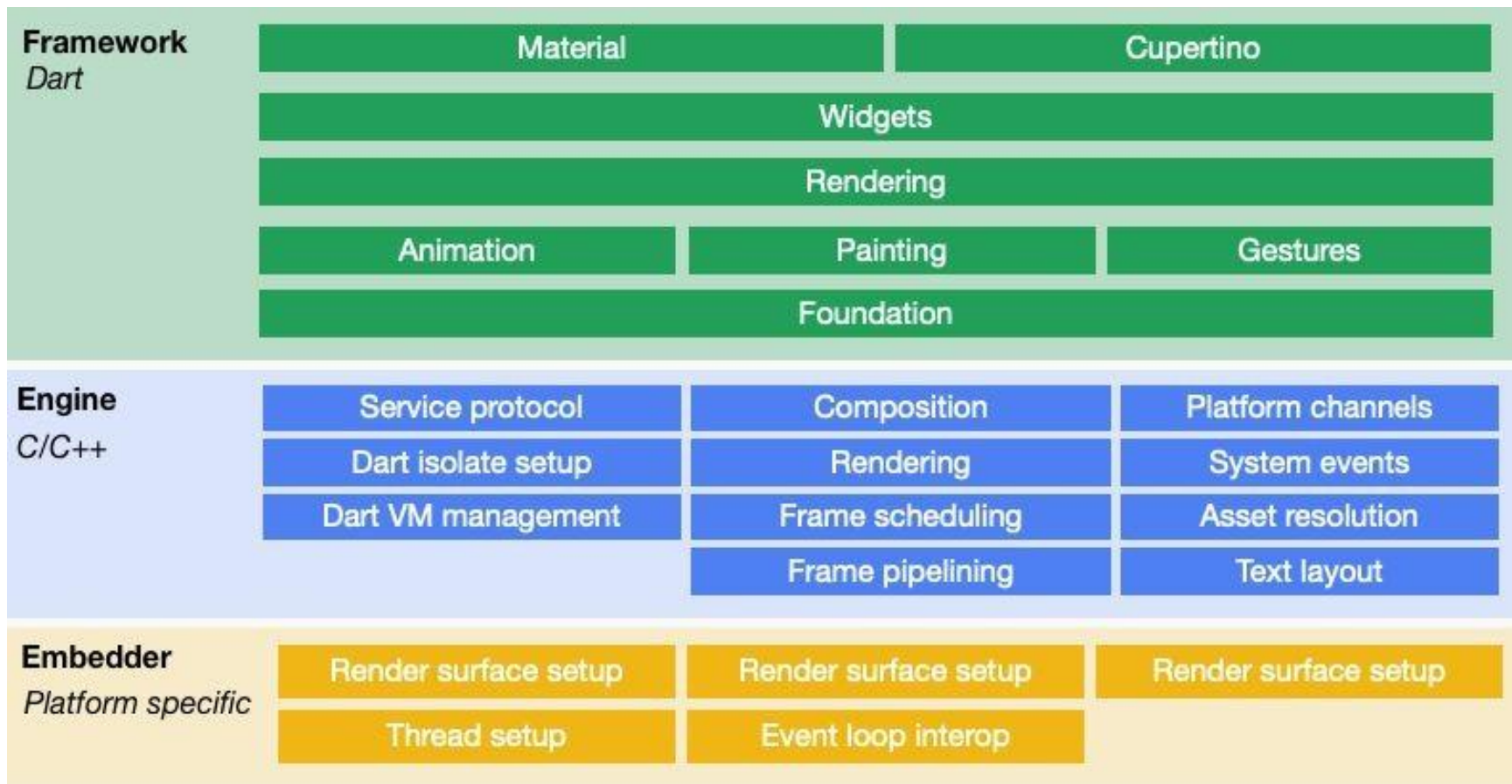
Reactive Views



Flutter Approach



Source : <https://medium.com/flutter-community/in-plain-english-so-what-the-heck-is-flutter-and-why-is-it-a-big-deal-7a6dc926b34a>



Source : <https://github.com/flutter/flutter/wiki/The-Engine-architecture>

4

Dart



► Langage Dart

- ▶ Développé par Google, 1ère version en 2011
- ▶ Initialement créé comme une **alternative à JavaScript**
- ▶ **Interprété et compilé**
- ▶ Compilation pour du **mobile, desktop, backend** ou en **JavaScript**

- ▶ **Langage fortement typé**
- ▶ **Langage orienté objet** : classe, héritage, interface, mixin, ...
- ▶ Permet des tâches asynchrones sur le même processus que l'UI
- ▶ Dart est single thread (=Isolate) par défaut mais il est possible de créer d'autres "Isolates"

Environnement Flutter / Dart

- ▶ Langage
- ▶ Environnements de développement intégré
- ▶ Hot reload
- ▶ Outils de débogage
- ▶ Tests
- ▶ CI / CD

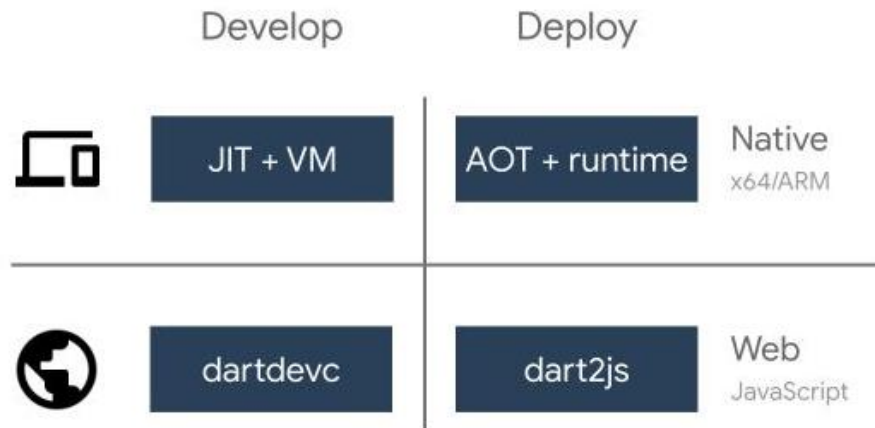
► Compilateur

Dart Native

- ▶ Cible les périphériques mobiles, desktop et serveurs
- ▶ Inclut une VM Dart avec compilateur JIT
- ▶ Inclut un compilateur AOT pour le code machine

Dart Web (JavaScript)

- ▶ Cible le web
- ▶ Inclut un compilateur pour le développement (dartdevc) et un compilateur pour la production



Source : <https://dart.dev>

Syntaxe facile à apprendre

Syntaxe comparable à d'autres langages tels que Kotlin, Swift ou TypeScript



Dart

```
class Segment {  
  int links = 4;  
  toString() => "I have $links links";  
}
```



Kotlin

```
class Segment {  
  var links: Int = 4  
  override fun toString()= "I have $links links"  
}
```



Swift

```
class Segment: CustomStringConvertible {  
  var links: Int = 4  
  public var description: String { return  
    "I have \(links) links"  
  }  
}
```



TypeScript

```
class Segment {  
  links: number = 4  
  public toString = () : string => { return  
    `I have ${this.links} links` };  
}
```

Exemple de programme

```
void main() {  
    print('Hello, World!');  
    var result = fibonacci(20);  
}  
  
int fibonacci(int n) {  
    if (n == 0 || n == 1) return n;  
    return fibonacci(n - 1) + fibonacci(n - 2);  
}
```

5

Installation de l'environnement Flutter

<https://flutter.dev/docs/get-started/install>

6

Ma 1ère application Flutter

Création d'un projet Flutter

► File → New → New Flutter Project

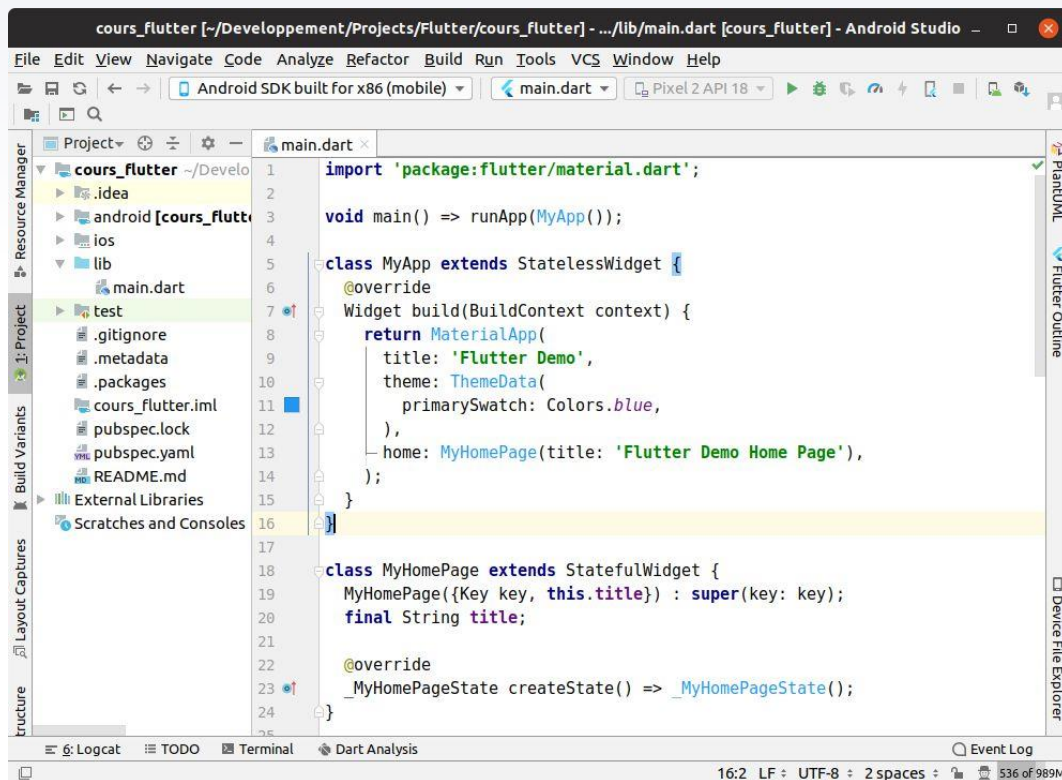
The image displays three sequential screenshots of the 'Create New Flutter Project' wizard in Android Studio.

Screenshot 1: New Flutter Project
This screen shows three options: 'Flutter Application' (selected), 'Flutter Plugin', and 'Flutter Package'. Below the options, instructions are provided: 'Select an "Application" when building for end users.', 'Select a "Plugin" when exposing an Android or iOS API for developers.', 'Select a "Package" when creating a pure Dart component, like a new Widget.', and 'Select a "Module" when creating a Flutter component to add to an Android or iOS app.'

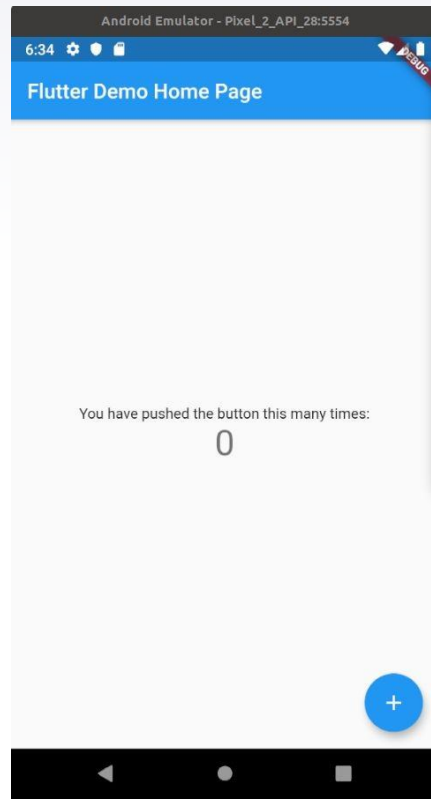
Screenshot 2: Configure the new Flutter application
This screen allows configuration of the new application. Fields include: 'Project name' (new_flutter_app), 'Flutter SDK path' (F:\Developpement\flutter), 'Project location' (F:\Developpement\Projects\Flutter\new_flutter_app), and 'Description' (Flutter c'est trop bien). A checkbox for 'Create project offline' is at the bottom right.

Screenshot 3: Set the package name
This screen is for setting the package name. It includes the instruction 'Applications and plugins need to generate platform-specific code'. The 'Package name' field is set to 'fr.company'. Under 'Platform channel language', both 'Include Kotlin support for Android code' and 'Include Swift support for iOS code' are checked.

Création d'un projet Flutter

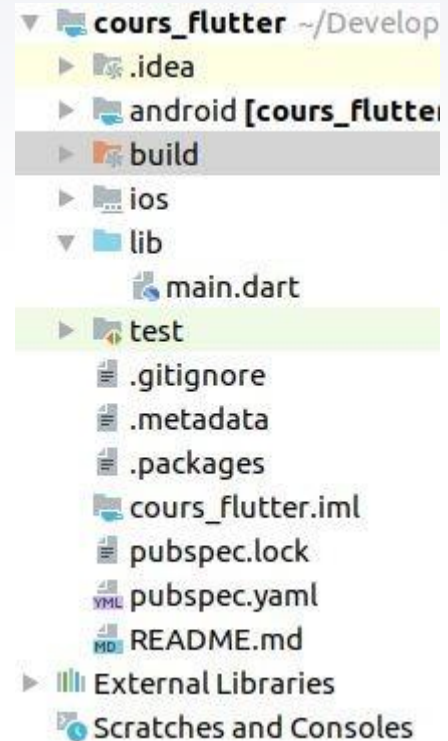


```
1 import 'package:flutter/material.dart';
2
3 void main() => runApp(MyApp());
4
5 class MyApp extends StatelessWidget {
6   @override
7   Widget build(BuildContext context) {
8     return MaterialApp(
9       title: 'Flutter Demo',
10      theme: ThemeData(
11        primarySwatch: Colors.blue,
12      ),
13      home: MyHomePage(title: 'Flutter Demo Home Page'),
14    );
15  }
16
17
18 class MyHomePage extends StatefulWidget {
19   MyHomePage({Key key, this.title}) : super(key: key);
20   final String title;
21
22   @override
23   _MyHomePageState createState() => _MyHomePageState();
24 }
```



Organisation d'un projet

- ▶ **android** : fichiers nécessaires pour la compilation d'une application Android
- ▶ **ios** : fichiers nécessaires pour la compilation d'une application IOS
- ▶ **lib** : sources du projet Flutter
- ▶ **test** : tests
- ▶ **pubspec.yaml** : gestion des dépendances



pubspec.yaml

- ▶ Gestion des dépendances
- ▶ Configuration du projet
- ▶ Définition des assets

Exécuter la commande suivante après chaque modification du fichier :

```
flutter pub get
```

```
name: cours_flutter
description: A new application.

version: 1.0.0+1

environment:
  sdk: ">=2.1.0 <3.0.0"

dependencies:
  flutter:
    sdk: flutter
  location: <=2.3.2
  http: ^0.12.0+2
  rxdart: ^0.22.2

dev_dependencies:
  flutter_test:
    sdk: flutter

flutter:
  uses-material-design: true
  assets:
    - assets/images/
```

Packages : pub.dev



Flutter Favorites

Packages that demonstrate the highest levels of quality, selected by the Flutter Ecosystem Committee

flutter_mobx

Flutter integration for MobX. It provides a set of Observer widgets that automatically

dart.pixelingene.com

flutter_bloc

Flutter Widgets that make it easy to implement the BLoC (Business Logic Component)

bloclibrary.dev

url_launcher

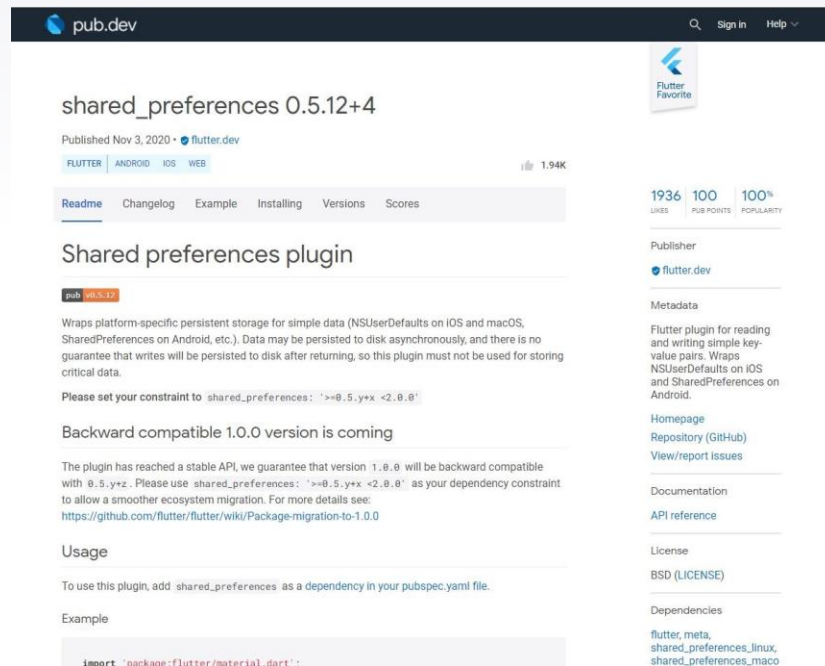
Flutter plugin for launching a URL on Android and iOS. Supports web, phone, SMS, and

flutter.dev

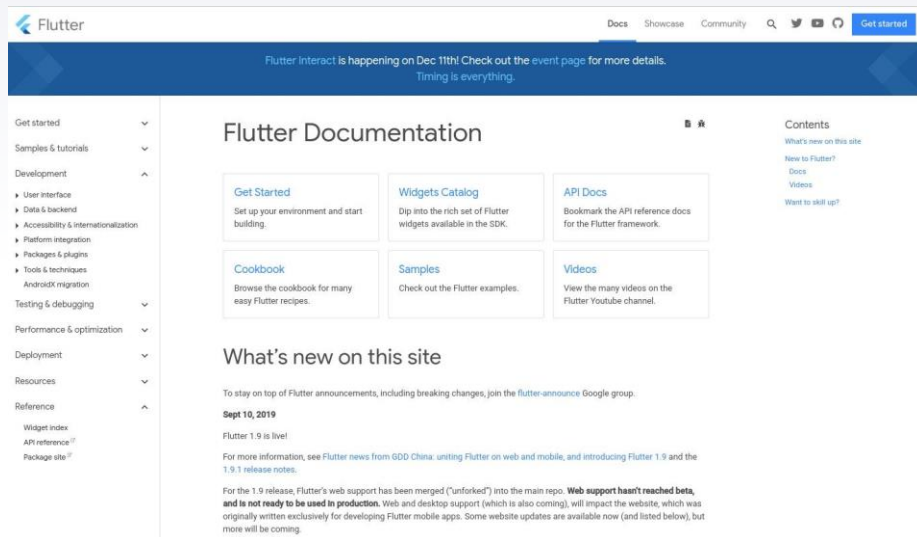
device_info

Flutter plugin providing detailed information about the device (make, model, etc.), and Android

flutter.dev

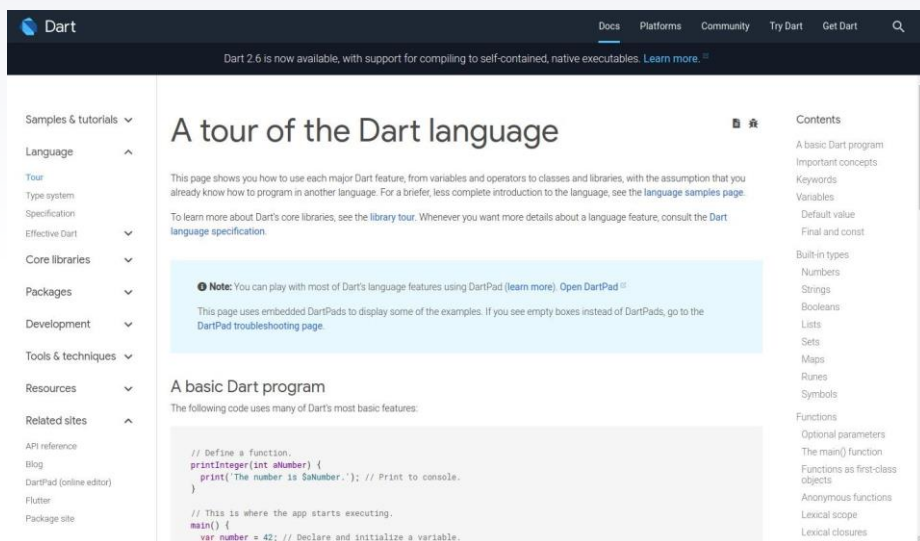


Documentation



The screenshot shows the Flutter Documentation website. The header includes the Flutter logo, navigation links (Docs, Showcase, Community, Get started), and a search bar. A blue banner at the top reads "Flutter Interact is happening on Dec 11th! Check out the event page for more details. Timing is everything." The main content area is titled "Flutter Documentation" and features a grid of links: "Get Started", "Widgets Catalog", "API Docs", "Cookbook", "Samples", and "Videos". A sidebar on the left lists various topics like "User interface", "Platform integration", and "Testing & debugging". A "What's new on this site" section is also visible, mentioning Flutter 1.9.

<https://flutter.dev>



The screenshot shows the Dart Documentation website. The header includes the Dart logo, navigation links (Docs, Platforms, Community, Try Dart, Get Dart), and a search bar. A dark banner at the top reads "Dart 2.6 is now available, with support for compiling to self-contained, native executables. Learn more." The main content area is titled "A tour of the Dart language" and includes a "Note" about using DartPad. A sidebar on the left lists various topics like "Language", "Core libraries", and "Development". A "What's new on this site" section is also visible, mentioning Dart 2.6.

<https://dart.dev>

Conventions de nommage

- ▶ **snake_case** (ex : "home_screen.dart")
 - ▶ noms de fichier
- ▶ **UpperCamelCase** (ex : "HomeScreen")
 - ▶ classes
 - ▶ typedef
 - ▶ constructeurs
- ▶ **lowerCamelCase** (ex : "toJson")
 - ▶ méthodes
 - ▶ variables
 - ▶ constantes

► Application Flutter

Fonction "main()" ⇒ point d'entrée de l'application
→ Fichier "lib/main.dart"

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


Hot reload

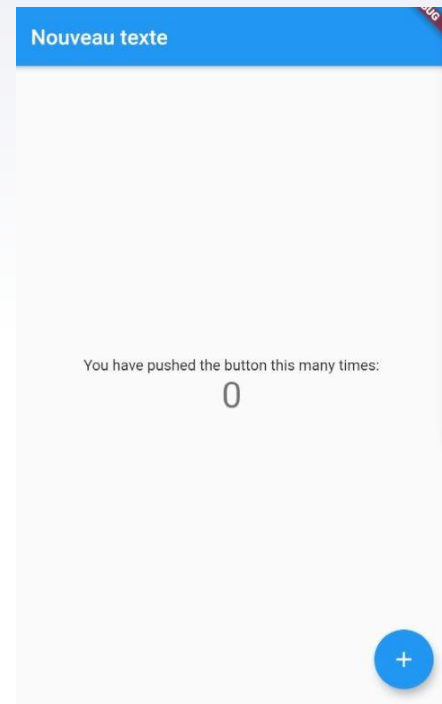
Dans le fichier "main.dart"

Remplacer le contenu de la ligne 29 :

```
home: MyHomePage(title: 'Flutter Demo Home Page'),
```

Par :

```
home: MyHomePage(title: 'Nouveau texte'),
```



A blue triangle pointing to the right, containing the number 7.

7

Flutter

Les Widgets

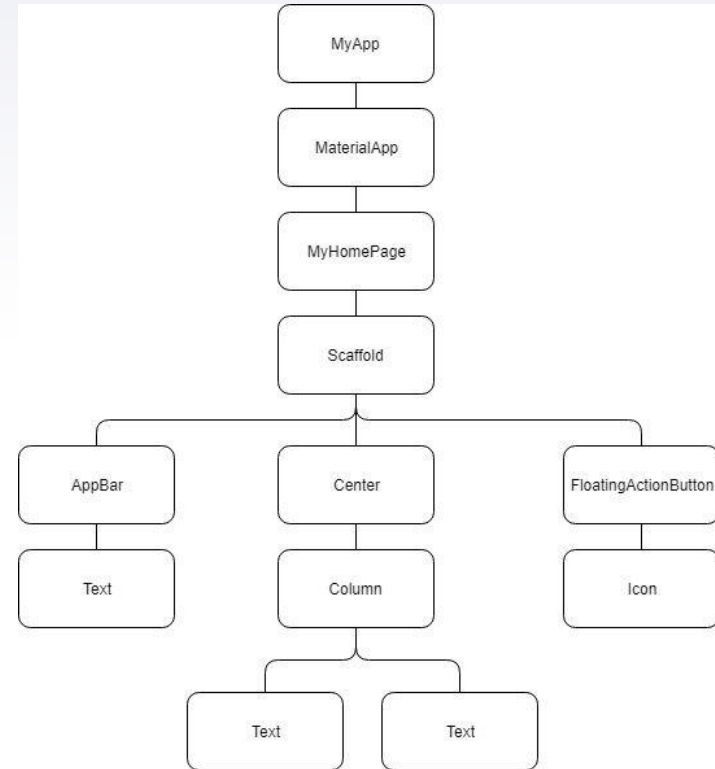
Widgets

"**StatelessWidget**", "**StatefulWidget**", "**State<T>**": quesako ?

```
class MyApp extends StatelessWidget {  
    [...]  
}  
  
class MyHomePage extends StatefulWidget {  
    [...]  
}  
  
class _ApplicationState extends State<Application> {  
    [...]  
}
```

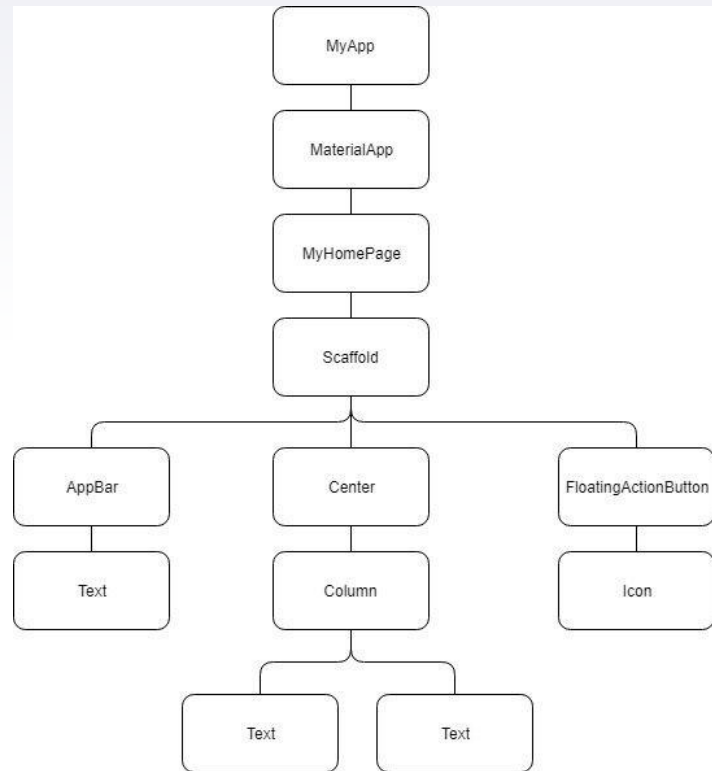
Widgets

- ▶ **Dans Flutter, tout est widget!**
- ▶ Le rôle principal d'un widget est de faire un rendu via la méthode "**build()**"
- ▶ Les widgets sont organisés sous forme **d'arborescence**
- ▶ La méthode "runApp" accepte uniquement un type Widget en paramètre



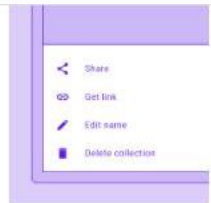
Widgets

```
@override
Widget build(BuildContext) {
  return Scaffold(
    appBar: AppBar(
      title: Text(widget.title),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Text('You have pushed the button...:'),
          Text('$_counter'),
        ],
      ),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: _incrementCounter,
      child: Icon(Icons.add),
    ),
  );
}
```



Widgets

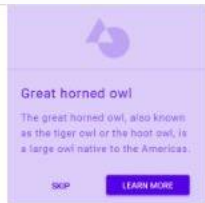
Flutter fournit une centaine de widgets de base :
<https://flutter.dev/docs/reference/widgets>



BottomSheet

Bottom sheets slide up from the bottom of the screen to reveal more content. You can call `showBottomSheet()` to implement a persistent bottom sheet or `showModalBottomSheet()` to implement a modal bottom sheet.

[Documentation](#)



ButtonBar

A horizontal arrangement of buttons.

[Documentation](#)



Card

A Material Design card. A card has slightly rounded corners and a shadow.

[Documentation](#)



RotationTransition

Animates the rotation of a widget.

[Documentation](#)



Row

Layout a list of child widgets in the horizontal direction.

[Documentation](#)



Scaffold

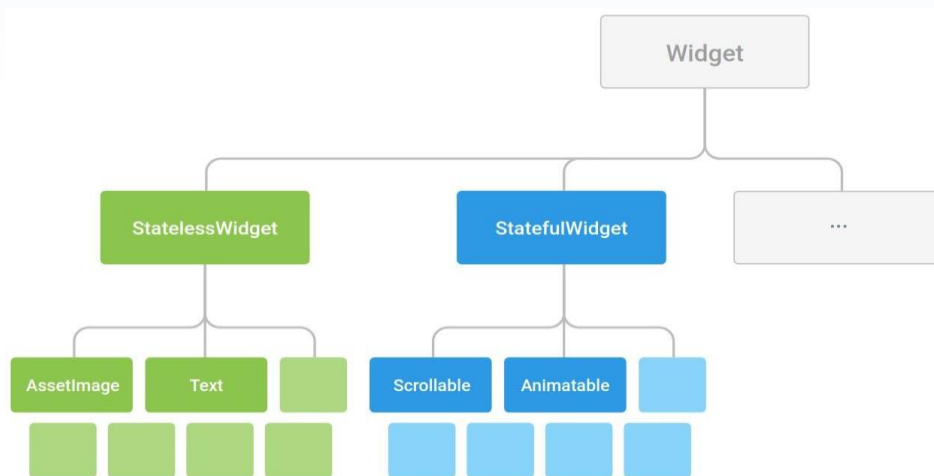
Implements the basic Material Design visual layout structure. This class provides APIs for showing drawers, snack bars, and bottom sheets.

[Documentation](#)

Widgets

3 grandes catégories de widgets :

- ▶ StatelessWidget
- ▶ StatefulWidget
- ▶ (InheritedWidget)



StatelessWidget

- ▶ **Pas d'état** (pas de state)
- ▶ Comporte une seule méthode : "**Widget build(BuildContext context)**"
- ▶ **Ne peut pas être "re-builder"** par lui-même
- ▶ Ex : **Icon, Text, Button,...**
- ▶ Cycle de vie :
 - ▶ Initialisation
 - ▶ Rendu (méthode build())

```
class MyApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return [...];  
  }  
}
```


StatefulWidget

- ▶ **Comporte un état**
- ▶ La méthode "build()" est déportée dans le state
- ▶ Une seule méthode : "**State<T> createState()**" où <T> correspond à la classe StatefulWidget
- ▶ Méthode "**build**" appelée à chaque **modification du state**
- ▶ State = **ensemble des attributs** de la classe State
- ▶ "**initState()**" : méthode pour initialisation le state

```
class MyApp extends StatefulWidget {  
  
    @override  
    _MyAppState createState() =>  
    _MyAppState();  
}  
  
class _MyAppState extends State<MyApp> {  
    int _counter;  
  
    @override  
    void initState() {  
        _counter = 0;  
        super.initState();  
    }  
  
    @override  
    Widget build(BuildContext context) {  
        return Text("Compteur $_counter");  
    }  
}
```

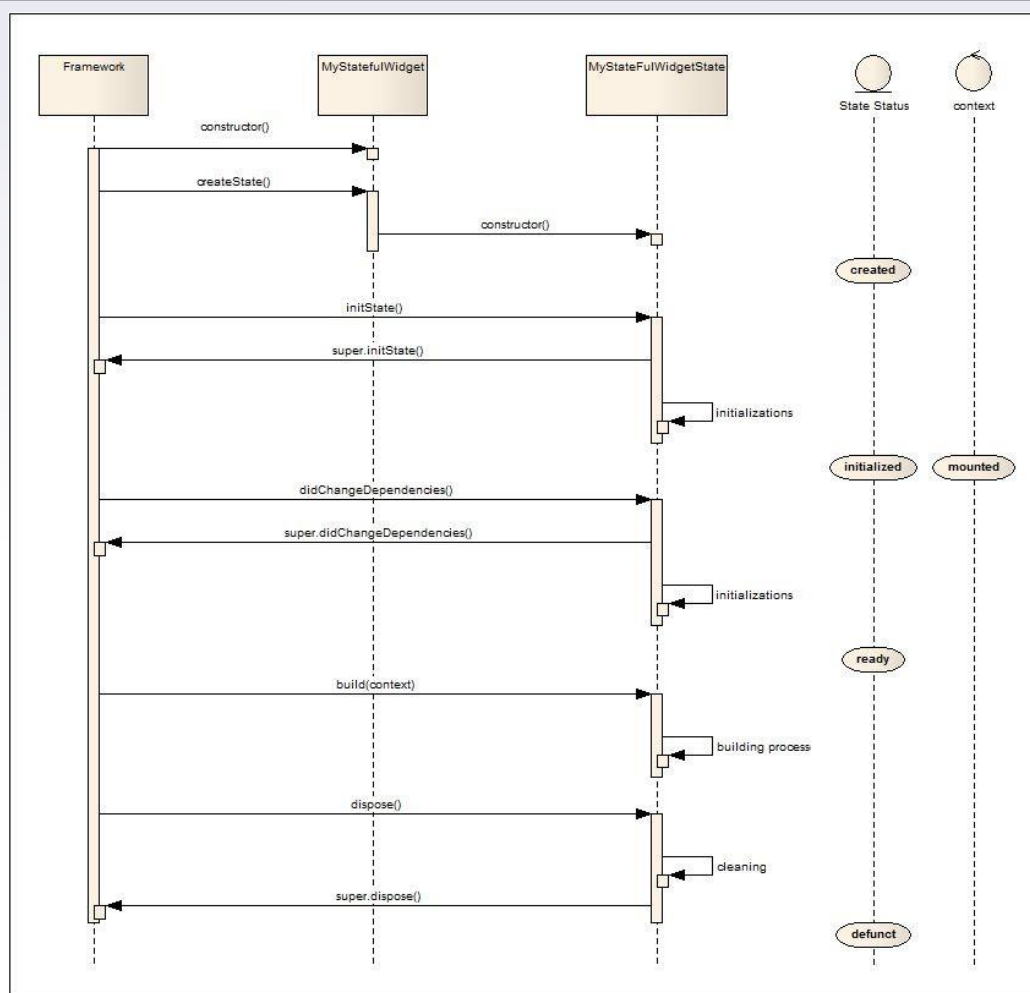
StatefulWidget

- ▶ Méthode "**setState**" pour modifier le state :

```
void setState(VoidCallback fn)
```

- ▶ La modification directement d'un attribut ne force pas l'appelle de la méthode "build"

```
class _MyAppState extends State<MyApp> {  
  int _counter = 0;  
  
  @override  
  Widget build(BuildContext context) {  
    return Column(  
      children: <Widget>[  
        Text("Compteur $_counter"),  
        RaisedButton(  
          onPressed: () {  
            setState(() {  
              _counter++;  
            });  
          },  
          child: Text("Compter")  
        )  
      ],  
    );  
  }  
}
```



Source : <https://www.didierboelens.com/fr/2018/06/widget---state---context---inheritedwidget/>

8

Dart

Les types

Types

- ▶ Tout ce qui est placé dans une variable **est un objet**, y compris les types "int", "double", "bool" et "null"

```
int a = 5;
```

- ▶ Si vous ne précisez pas un type à une variable, Dart va automatiquement le déduire

```
var a = 5;
```

- ▶ Le type **"dynamic"** équivaut à tout type

```
dynamic a = 5;  
a = "toto";
```

Types

▶ Nombres :

- ▶ int
- ▶ double
- ▶ num

```
int num1 = 3;  
double num3 = 3.1;  
num num5 = 3;  
num num7 = 3.1;
```

"int" et "double" héritent de "num"

▶ Booleans :

```
bool val = true;
```

▶ String :

```
String s1 = 'chaine1';  
var s2 = "chaine2";  
String s3 = '''  
    ligne1  
    ligne2  
    ligne3  
    ''';
```

```
String name = 'toto';  
String s1 = 'Name: $name';  
String s2 = 'Name: ${name.toLowerCase()}';
```

Types

► List :

Pas de type "Array" dans Dart

```
List<int> liste1      = List();
List<int> liste2      = [];

List<int> liste3      = [1, 2, 3];

List<dynamic> liste4 = [1, 'Hello', 3.3];
List liste5          = [4, "ma liste"];

print(liste3[0]);    // Affiche 1

liste3.add(4);
print(liste3.length); // Affiche 4
```

► Map :

```
Map<String, int> map1 = Map();
Map<String, int> map2 = {'var1': 1, 'var2': 2};
print(map2['var1']); // Affiche 1

Map<String, dynamic> map3 = Map();
map3['toto'] = 'titi';
map3['tata'] = 3;

Map map4 = Map(); // <=> Map<dynamic, dynamic>
map4['toto'] = 'titi';
map4[4] = 8;
print(map4[4]); // Affiche 8
```

9

Dart

Programmation objet (1/2)

POO

Dart reprend les concepts de la programmation objet :

- ▶ constructeur
- ▶ public / private
- ▶ héritage
- ▶ interface
- ▶ classe abstraite
- ▶ ...

POO

```
class Person {  
    String name;  
    Person(this.name);  
}
```

Constructeur équivalent à :

```
Person(String name) {  
    this.name = name;  
}
```

```
Person p = Person("toto");
```

POO

- ▶ Dart n'utilise pas les termes "public", "protected" et "private"
- ▶ Par défaut un objet est public.
- ▶ Pour le rendre privée il faut le **préfixer par un "_"**

```
int _counter = 0;
```

Getters & Setters

- ▶ Chaque attribut public possède un "getter" et un "setter" implicites

```
class Square {  
  num side;  
  Square(this.side);  
}
```

```
var s = Square(4);  
s.side = 5;  
print(s.side);
```

- ▶ Il est possible d'ajouter ses propres getters et setters :

```
class Square {  
  num side;  
  Square(this.side);  
  
  num get perimetre => this.side * 4;  
  set perimetre(num value) {  
    this.side = value / 4;  
  }  
}
```

```
var s = Square(4);  
print(s.perimetre);  
s.perimetre = 12;
```

Héritage

- ▶ Utilisation du mot clé "**extends**"
- ▶ Appelle du constructeur parent avec le mot clé "**super**"
- ▶ L'héritage multiple n'est pas possible =>**utiliser Mixin**

```
class Person {  
    String name;  
    Person(this.name);  
}
```

```
class Student extends Person {  
    int studentId;  
    Student(String name, this.studentId) : super(name);  
}
```

10

Dart

Structures de contrôle & opérateurs

Structures de contrôle

```
if (year < 1901) {  
    print('19st century');  
} else if (year < 2001) {  
    print('20th century');  
} else {  
    print('21th century');  
}
```

```
String visibility = isPublic ? 'public' : 'private';
```

Structures de contrôle

```
List<String> students = ['s1', 's2', 's3'];  
for (var student in students) {  
    print(student);  
}  
  
for (var i = 0; i < students.length; i++) {  
    print(students[i]);  
}  
  
while (year < 2016) {  
    year += 1;  
}
```


Structures de contrôle

- ▶ Avec une fonction lambda (ou anonyme) :

```
// foreach((String) → void f) → void
students.forEach((student) {
    print(student);
});
```

```
students.asMap().forEach((index, student) {
    print('$index: $student');
});
```

11

Dart

Les paramètres



Paramètres d'une fonction

```
void printMsg(String msg, bool bold, Color color) {  
    [...]  
}
```

```
printMsg("texte à afficher", false, Colors.red);
```

Paramètres optionnels

- ▶ Les paramètres optionnels s'utilisent avec les crochets :

```
void printMsg(String msg, [bool bold, Color color]) {  
    [...]  
}  
  
printMsg("texte à afficher", false, Colors.red);  
printMsg("texte à afficher", false);  
printMsg("texte à afficher");
```

- ▶ Si la paramètre n'est pas renseigné, alors celui-ci **=null**

```
void printMsg(String msg, [bool bold = false, Color color = Colors.black]) {  
    [...]  
}
```

Paramètres nommés

- ▶ Les paramètres nommés s'utilisent avec les accolades :

```
void printMsg({String msg, bool bold, Color color}) {  
    [...]  
}  
  
printMsg(  
    msg: "texte à afficher",  
    bold: false,  
    color: Colors.red  
);
```

Paramètres nommés

- ▶ Pour rendre un paramètre nommé obligatoire, il faut ajouter l'annotation **@required**:

```
void printMsg(@required String msg, bool bold, Color color) {...}
```

- ▶ Possibilité de combiner les paramètres nommés avec les non-nommés :

```
void printMsg(String msg, {bool bold, Color color = Colors.red}){...}  
  
printMsg("texte à afficher",  
    bold: true,  
    color: Colors.red  
);
```

12

Flutter

La navigation

Navigation

2 versions du Navigator :

- ▶ **Navigator 1.0**, basé sur :
 - ▶ Navigator
 - ▶ Routes (routes nommées et anonymes)
- ▶ **Navigator 2.0**, basé sur :
 - ▶ Page
 - ▶ Router

Pour une application mobile, Navigator 1.0 est plus simple à utiliser.
Navigator 2.0 intéressant pour du web.

Navigation

- ▶ Route anonyme

```
Navigator.of(context).push(MaterialPageRoute(builder: (context) => MyScreen()));
```

- ▶ "Navigator.push" avec valeur de retour

```
final String result = await Navigator.of(context).push(  
    MaterialPageRoute(builder: (context) => MyScreen())  
);
```

- ▶ "Navigator.pop" => dépile un écran

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

Navigation

- ▶ Il est possible de définir des routes nommées :

```
Navigator.of(context).pushNamed('/detail');
```

- ▶ Nécessite de déclarer les routes dans "MaterialApp" :

```
class MyApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      title: 'Flutter Demo',  
      routes: {  
        '/home': (context) => Home(),  
        '/detail': (context) => Detail(), // Vous pouvez utiliser des constantes  
      },  
      home: Home(),  
    );  
  }  
}
```

Navigation

- ▶ Envoie des paramètres à l'écran :

```
Navigator.of(context).pushNamed(  
  'detail',  
  arguments: {  
    'companyId': companyId  
  }  
);
```

- ▶ Récupération des paramètres :

```
final Map<String, dynamic> args = ModalRoute.of(context).settings.arguments;  
final int companyId = args['companyId'];
```

13

Dart

Programmation objet (2/2)

Constructeurs nommés

```
class Person {  
    String name;  
    Person.firstConstructor(this.name);  
    Person.secondConstructor() {  
        this.name = "toto";  
    }  
}  
  
void main() {  
    Person p = new Person.firstConstructor("toto");  
    Person p2 = new Person.secondConstructor();  
}
```

Constructeurs nommés

- ▶ Il ne peut y avoir qu'un seul constructeur "non nommé"

```
class Person {  
    String name;  
    int age;  
  
    Person(this.name, this.age);  
    Person.name(String name): this(name, 0);  
}  
  
void main() {  
    Person p = Person.name("toto");  
}
```

Factory

```
void main() {  
    Person p = new Person();  
}  
  
class Person {  
    static final Person _instance = Person._internal();  
  
    factory Person() => _instance;  
    Person._internal(); // Constructeur privé  
}
```

Classes abstraites

- Utilisation du mot clé **"abstract"**

```
abstract class Repository {  
    List<User> getUsers();  
}  
  
class LocalRepository extends Repository {  
    List<User> getUsers() {  
        [...]  
    }  
}
```


Interface

- ▶ Toute classe peut être considérée comme interface
- ▶ Il est possible d'implémenter **plusieurs interfaces**

```
class A {  
    String testA() => "a";  
}  
  
abstract class B {  
    String testB();  
}
```

```
class Ab implements A, B {  
    String testA() => "ab_a";  
    String testB() => "ab_b";  
}
```



Dart

Programmation asynchrone

Programmation asynchrone

- ▶ Par défaut, le code Dart est exécuté sur **1 seul thread**.
- ▶ Tout code bloquant le thread bloque le programme.
- ▶ Dart propose des opérateurs pour pouvoir **exécuter du code asynchrone**.

Fonction asynchrone:

- ▶ "Future<T>" représente le **résultat d'un traitement asynchrone**.
- ▶ "**T**" étant le type retourné par le traitement.

```
Future<Response> getRequest(String url) {  
    Response response;  
    [...]  
    return response;  
}
```

Un type "Future" peut être soit "**complet**", soit "**incomplet**".

Si une fonction asynchrone ne renvoie pas de valeur son type doit être "**Future<void>**".

► Programmation asynchrone

- ▶ Lorsqu'on appelle une fonction asynchrone, celle-ci met **en file d'attente** le travail à effectuer et renvoie un futur "incomplet".
- ▶ Quand une opération asynchrone se termine, la valeur "Future" prend une valeur ou une erreur.

```
final Future<Response> response = client.getRequest(uri);
```

▶ Programmation asynchrone

- ▶ La méthode "**then**" (callback) permet de récupérer le résultat de la fonction quand elle se termine ou tombe en erreur :

```
final Future<Response> response = client.getRequest(uri);  
response.then((value) {  
    print('ok');  
}, onError: (e) {  
    print('error: $e');  
});
```

- ▶ Le paramètre "**onError**" est **optionnel** dans la fonction "then"

Programmation asynchrone

- ▶ Le mot clé "**await**" permet de récupérer directement la valeur de la fonction asynchrone sans passer par la fonction "then"
- ▶ Avec await, une fonction retourne le type directement et non un Future

```
final Response response = await client.getRequest(uri);
```

- ▶ "await" est **bloquant** et ne peut donc être utilisé que sur une fonction également **asynchrone** possédant le mot clé "**async**"

```
Future<List<User>> getUsers() async {  
    List<User> users = [];  
    Response response = await client.getRequest(uri);  
    [...]  
    return users;  
}
```

15

Dart

Null safety

▶ Null safety

- ▶ Disponible avec **Flutter 2** et **Dart 2.12**
- ▶ Pour activer "null safety" au sein du projet :

```
environment:  
  sdk: ">=2.12.0 <3.0.0"
```

- ▶ Pour migrer : <https://dart.dev/null-safety/migration-guide>

Null safety

Avec "null safety" activé :

- ▶ Tous les objets sont par défaut "non-nullable", ils ne peuvent pas recevoir la valeur null
- ▶ Pour pouvoir recevoir la valeur "null", il faut indiquer "?" après le type
- ▶ Un objet "non-nullable" ne peut pas recevoir la valeur d'un objet "nullable" même si celui-ci n'est pas null

```
void main() {  
    final List<String?> liste = ["toto", "titi", null];  
    final int n = new Random().nextInt(3);  
    final String value = liste[n];  
}
```

Error: A value of type 'String?' can't be assigned to a variable of type 'String' because 'String?' is nullable and 'String' isn't.

```
    final String value = liste[n];  
                        ^
```

Error: Compilation failed.

Opérateurs Null-Safe

Il existe plusieurs opérateurs "null-safe" pour éviter les **NullPointerException** dans le cas d'un objet "nullable" :

- ▶ L'opérateur "?" permet d'accéder à un élément d'un objet ayant potentiellement

```
person?.name?.toLowerCase();
```

- ▶ L'opérateur "??" permet d'utiliser une valeur par défaut dans le cas où l'instruction est null :

```
person?.name?.toLowerCase() ?? 'Unknown';
```

- ▶ L'opérateur "??=" permet d'assigner une valeur à une variable si celle-ci est null :

```
String s = map[0] ;  
s ??= ' ';
```

16

Flutter

State Management

State Management

Différentes approches (complémentaires)

Gestion des données :

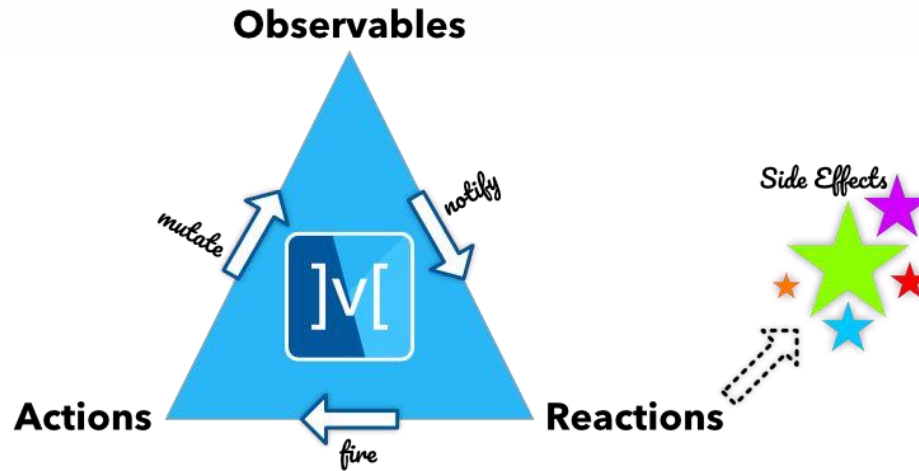
- ▶ State local (StatefulWidget)
- ▶ MobX
- ▶ BLoC / Rx
- ▶ Redux

Accès aux ressources :

- ▶ InheritedWidget
- ▶ Provider
- ▶ GetX

MobX

- ▶ Issue du monde JavaScript (mobx.js.org)
- ▶ Basé sur pattern "Observer"



Source : <https://github.com/mobxjs/mobx.dart>

MobX

- ▶ **Observables** : state de l'application
Peut être un objet simple (int, String, ...) ou complexe (liste, classe, ...)
- ▶ **Computed Observables** : données calculées basées sur les observables
- ▶ **Actions** : manipulation des observables. S'occupent de notifier les observateurs
- ▶ **Reactions** : observent les observables et reçoivent une notification quand un observable qu'ils suivent est modifié

MobX

```
part 'counter_store.g.dart';

class CounterStore = _CounterStore with
  _$CounterStore;

abstract class _CounterStore with Store {

  @observable
  int counter = 0;

  @action
  void increment() {
    counter++;
  }
}
```

```
class CounterExample extends StatelessWidget {
  final CounterStore counterStore = CounterStore();

  @override
  Widget build(BuildContext context) => Scaffold(
    appBar: AppBar(
      backgroundColor: Colors.blue,
      title: const Text('MobX Counter'),
    ),
    body: Center(
      child: Column(
        children: <Widget>[
          const Text('You have pushed the button...'),
          Observer(
            builder: (context) {
              return Text('${counterStore.counter}');
            }
          ),
        ],
      ),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: () => counterStore.increment(),
      child: const Icon(Icons.add),
    ),
  );
}
```



MobX

pubspec.yaml

```
dependencies:  
  mobx  
  flutter_mobx  
  
dev_dependencies:  
  build_runner  
  mobx_codegen
```

Commande à exécuter pour générer les fichiers "*.g.dart" :

```
flutter packages pub run build_runner build
```




MobX

Avantages :

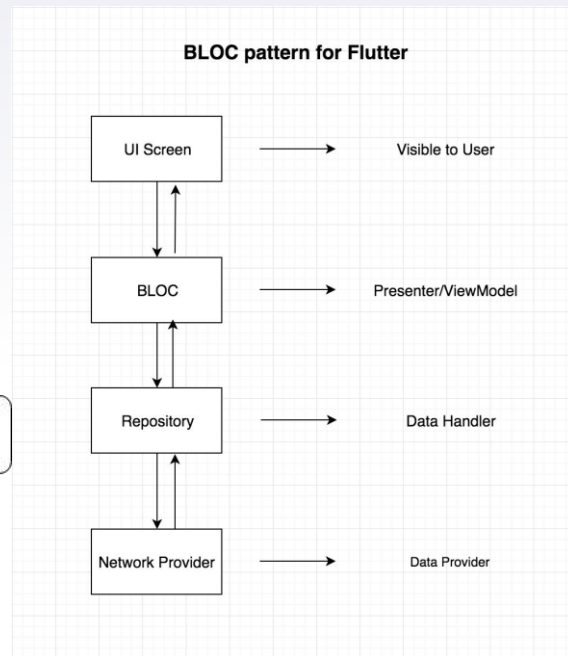
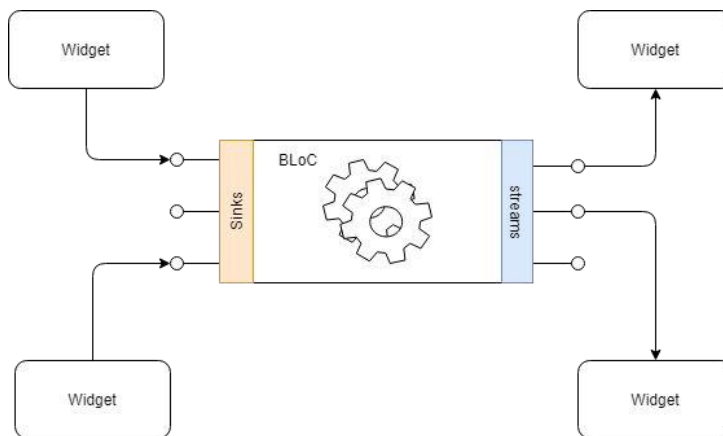
- ▶ Lisibilité du code, tant du côté "Store" que du côté UI
- ▶ Simple à mettre en oeuvre

Inconvénients :

- ▶ Oblige la génération de code (fichier *.g.dart)

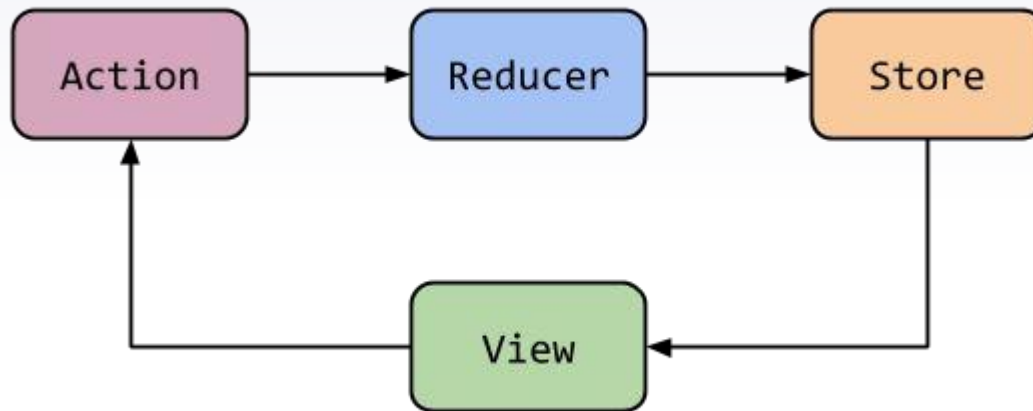
BLoC

- ▶ Business LogicComponent
- ▶ Basé sur pattern "Observer"
- ▶ Utilisation des "Stream"
- ▶ Plusieurs écoles :
 - ▶ 1bloc parécran
 - ▶ 1bloc parmétier



Source : <https://www.didierboelens.com/fr/2018/08/reactive-programming---streams---bloc/>

▶ Redux



Source : <https://blog.novoda.com/introduction-to-redux-in-flutter/>

17

Flutter

Tests

Tests

- ▶ **Tests unitaires** : tester une fonction, une classe
- ▶ **Tests de widget** : tester un widget
- ▶ **Tests d'intégration** : tester un parcours utilisateur au sein de l'application

Tests unitaires

- ▶ Les fichiers de tests doivent se trouver dans le **répertoire "test"** et **se terminer par "_test.dart"** afin qu'ils soient détectés en tant que fichiers de test
- ▶ Possibilité de regrouper les tests dans une fonction "group"

```
class Counter {  
  int value = 0;  
  
  void increment() => value++;  
  
  void decrement() => value--;  
}
```

```
// Import the test package and Counter class  
import 'package:test/test.dart';  
import 'package:counter_app/counter.dart';  
  
void main() {  
  test('Counter value should be incremented', () {  
    final counter = Counter();  
  
    counter.increment();  
  
    expect(counter.value, 1);  
  });  
}
```

Tests unitaires

Pour simuler un service web ou une base de données, il est possible de "simuler" ces dépendances via le plugin "Mockito".

```
class MockClient extends Mock implements http.Client {}

main() {
  group('fetchPost', () {
    test('returns a Post if the http call completes successfully', () async {
      final client = MockClient();

      when(client.get('https://jsonplaceholder.typicode.com/posts/1'))
        .thenAnswer((_) async => http.Response('{"title": "Test"}', 200));

      expect(await fetchPost(client), const TypeMatcher<Post>());
    });
  });
}
```

Tests de widget

Tester unitairement chaque widget :

```
void main() {  
  testWidgets('MyWidget has a title and message', (WidgetTester tester) async {  
    await tester.pumpWidget(MyWidget(title: 'T', message: 'M'));  
    final titleFinder = find.text('T');  
    final messageFinder = find.text('M');  
  
    // Use the `findsOneWidget` matcher provided by flutter_test to verify  
    // that the Text widgets appear exactly once in the widget tree.  
    expect(titleFinder, findsOneWidget);  
    expect(messageFinder, findsOneWidget);  
  });  
}
```


Tests d'intégration

- ▶ Tester l'enchaînement des écrans
- ▶ Tester les interactions entre les widgets
- ▶ Vérifier les performances

```
test('starts at 0', () async {  
  // Use the `driver.getText` method to verify the counter starts at 0.  
  expect(await driver.getText(counterTextFinder), "0");  
});  
  
test('increments the counter', () async {  
  // First, tap the button.  
  await driver.tap(buttonFinder);  
  
  // Then, verify the counter text is incremented by 1.  
  expect(await driver.getText(counterTextFinder), "1");  
});
```

Tests d'intégration

- ▶ Les tests d'intégration doivent se situer dans le fichier "integration_test/integration_test.dart"
- ▶ Commande à exécuter pour lancer les tests d'intégration :

```
flutter drive --target=integration_test/integration_test.dart
```

18

Best practices

Best practices

- ▶ Utiliser Dart Analysis pour conserver un code propre
- ▶ Respecter au maximum le "flux" de la "clean architecture"
- ▶ Utiliser "Statefull" uniquement si besoin de state local, sinon "Stateless"
- ▶ Préférer décomposer en sous-widgets plutôt que de passer par une méthode renvoyant un widget
- ▶ Découper les écrans en petits widgets réutilisables et testables
- ▶ Minimiser le "métier" dans l'UI -> services