Développement Multiplateforme

Flutter

Développement Multiplateforme

Flutter

- Introduction
- Installation
- Les bases
- Navigation
- Librairie externe
- Redux
- Réseau
- Animation

ENVOXOO

Introduction

Flutter est un framework de développement d'applications mobiles / desktop / web

Proposé en Open Source par Google

Moyen unifié de développer des applications pour Android / IOS / web / appli desktop à partir d'un code commun

Simplifie la gestion de compatibilité entre version d'OS

Introduction

Google met en avant les points suivants:

- Haute vitesse de développement (hot reload, collection de widget, plugin IDE)
- Design de qualité et personnalisable (widget, bibliothèque d'animation, architecture en couches extensibles)
- Expérience de haute qualité (puissance du moteur et interopérabilité)

Introduction

Flutter s’appuie sur le langage Dart qui est propulsé par Google

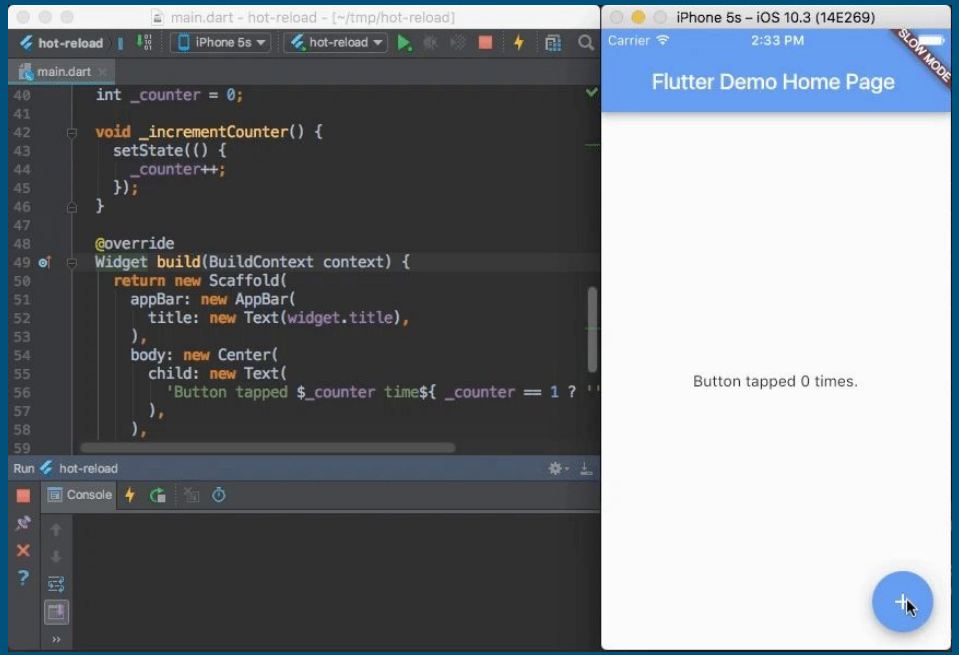
Même moteur de rendu que Chrome (Skia Graphics Library)

Ce moteur est très important car Flutter dessine chaque composant sans s’appuyer sur les composants du système

Flutter propose des répliques des composants systèmes pour garder une expérience native

Introduction

Google met à disposition des plugins pour plusieurs IDE (IntelliJ, Android Studio, VS Code)



Introduction

En quoi Flutter est différent (meilleur) que les autres?

Le développement mobile est assez récent, il est donc normal que les outils évoluent encore beaucoup…

Petite histoire du développement mobile...

Introduction

Les SDK IOS et Android

2008 pour IOS et 2009 pour Android basé sur Objective-C et Java

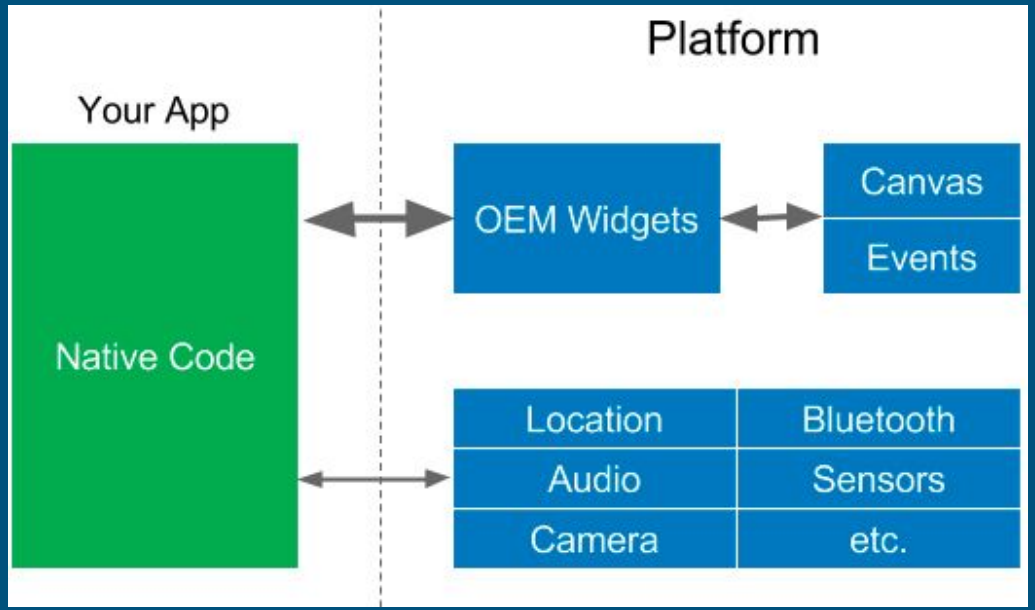
L'application communique avec la plateforme pour créer des widgets ou accéder à des services

Le rendu des widget est effectué sur un Canvas et les événements sont renvoyés aux widgets

=> Nécessité de créer une application pour chaque plateforme

Introduction

Les SDK IOS et Android



Introduction

Les WebViews

Solution basée sur le javascript et les webviews (phonegap, cordova, ionic…)

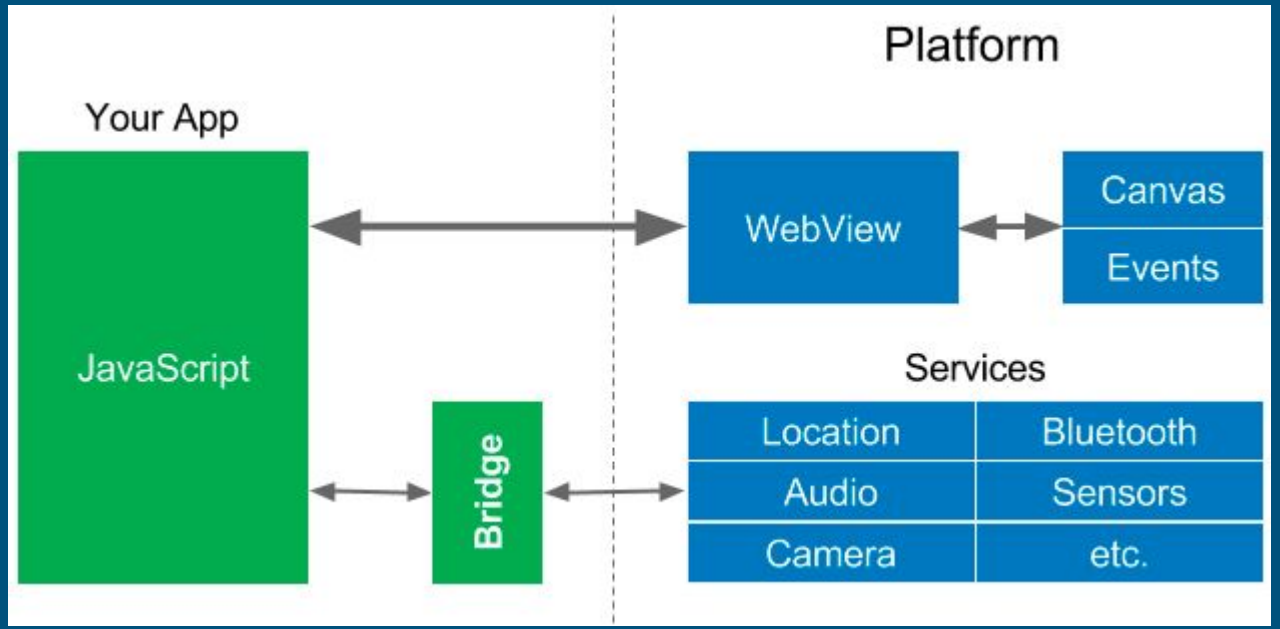
L'application génère du HTML et l'affiche dans une webview

Un "pont" est nécessaire pour utiliser les services du système

Plus on utilise le bridge, plus les performances se dégradent

Introduction

Les WebViews



Introduction

Les applications réactives

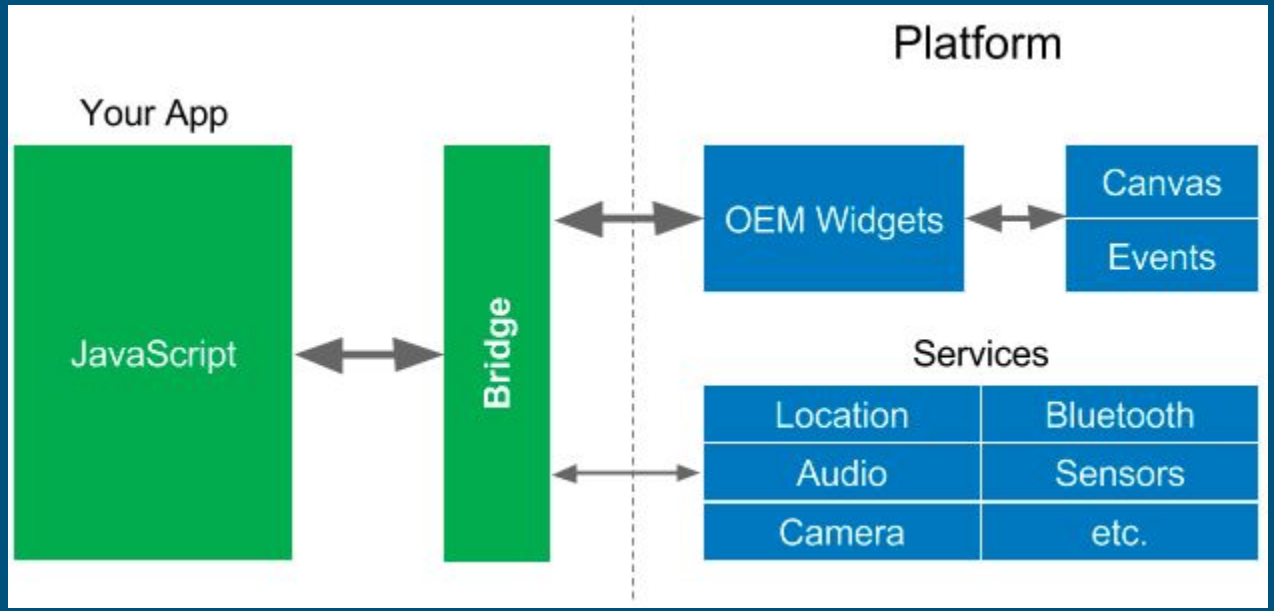
Solution basée sur le javascript et l'utilisation de composants natifs

L'utilisation d'un pont permet d'utiliser les composants natifs et les services du système

Amélioration des performances de façon générale

Introduction

Les applications réactives



Introduction

Flutter

Flutter propose des vues dynamiques comme React Native

L'approche est néanmoins différente afin de réduire les problèmes de performance liés au bridge

Flutter utilise Dart qui est un langage compilé

Introduction

Flutter

Le code Dart est donc compilé en code natif

Flutter peut donc communiquer directement avec le système (sans bridge)

Le code compilé permet aussi de réduire le temps de démarrage de l’application

Introduction

Flutter

Les **Widgets** sont des éléments qui constituent l'interface d'une application

Flutter n'utilise pas les composants du téléphone ou du web

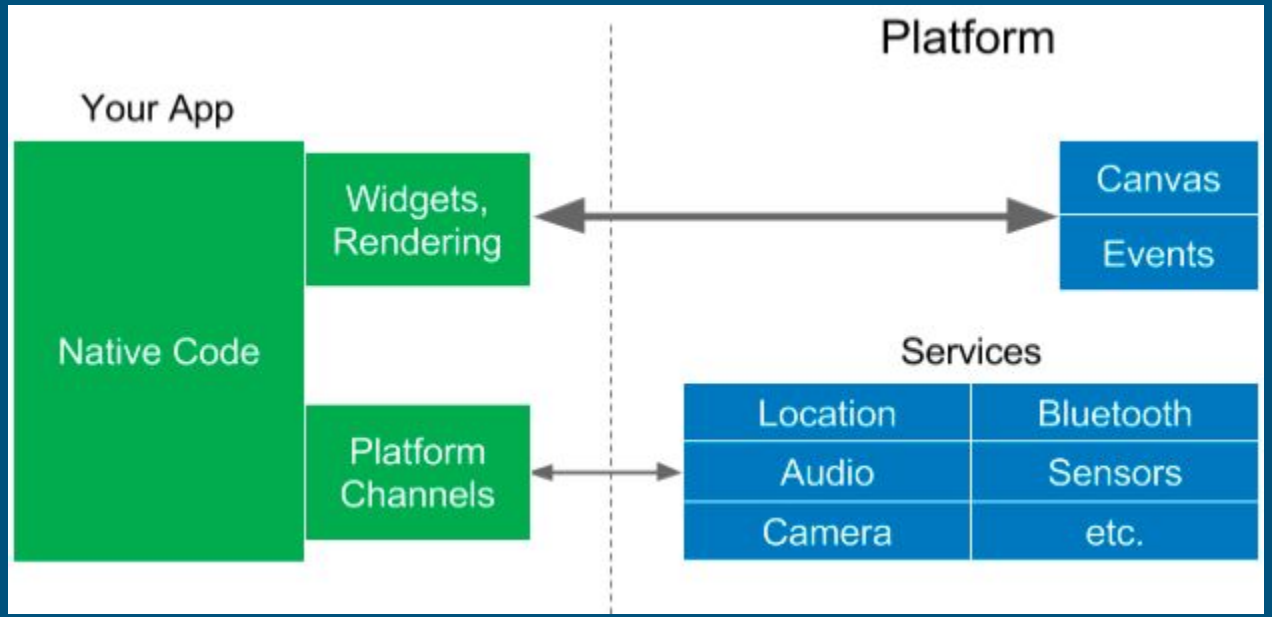
Flutter propose une série de widgets rapides, beaux et extensibles

Flutter déplace les widgets et le rendu dans l'application (Canvas)

L'interface entre Dart et la plateforme est beaucoup plus rapide que les bridges

Introduction

Flutter



Introduction

Positionnement

A la base Flutter est une expérimentation chez Google (équipe Chrome) afin de créer un moteur de rendu plus rapide

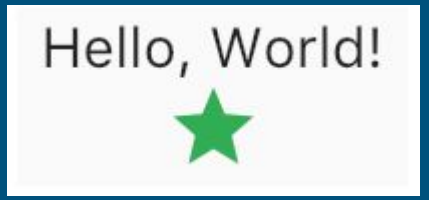
Chaque Widget a un nombre réduit de règle de positionnement

Tout ce qui est manipulé est un Widget

Introduction

Positionnement

```
new Center(
  child: new Column(
    children: [
      new Text('Hello, World!')),
      new Icon(Icons.star, color: Colors.green)
    ]
  )
)
```



Introduction

Compatibilité

Les widgets et le moteur de rendu font partie de l’application, il n’y a pas besoin de bibliothèques de compatibilité pour s’adapter à certaines plateformes

Les applications fonctionneront de la même manière quelque soit la version des OS

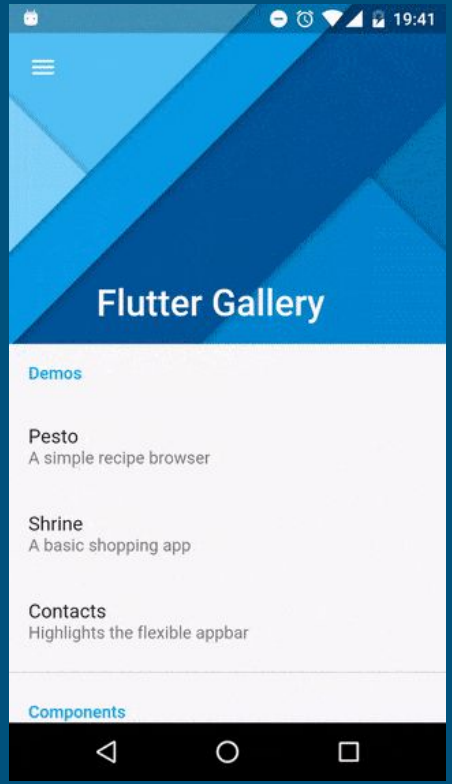
Introduction

Autres avantages

Dart possède un dépôt de paquets logiciels (pub)

Flutter est open source, même si Google ne le maintient pas assez rapidement, on peut toujours le faire

Introduction



ENVOXOO

Dart

Prenez quelques minutes pour parcourir l’introduction au langage Dart:

https://dart.dev/language

Installation

Suivez le guide selon votre OS à l'adresse ci-dessous:

https://flutter.io/get-started/install/

Installation

On termine l’installation en installant deux plugins pour IntelliJ

https://docs.flutter.dev/get-started/editor

Les bases

Voici le code minimal d'une application

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

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

Les bases

Flutter possède des composants de base, par exemple

Text

Row, Column

Container

On peut aussi utiliser des composants spécifiques (MaterialDesign / Cupertino)

Exemples de code sur https://flutter.io/widgets-intro/

Dart

Suivez le premier code Lab ci-dessous:

https://codelabs.developers.google.com/codelabs/flutter-codelab-first

Dart

Suivez le mini guide Flutter pour vous familiariser avec les composants et les concepts de base:

https://github.com/antz22/ultimate-guide-to-flutter

Les bases

Notion de Widgets

Presque tout est Widget

Un Widget peut être associé à un composant visuel ou un composant qui interagit avec l’aspect visuel de l’application

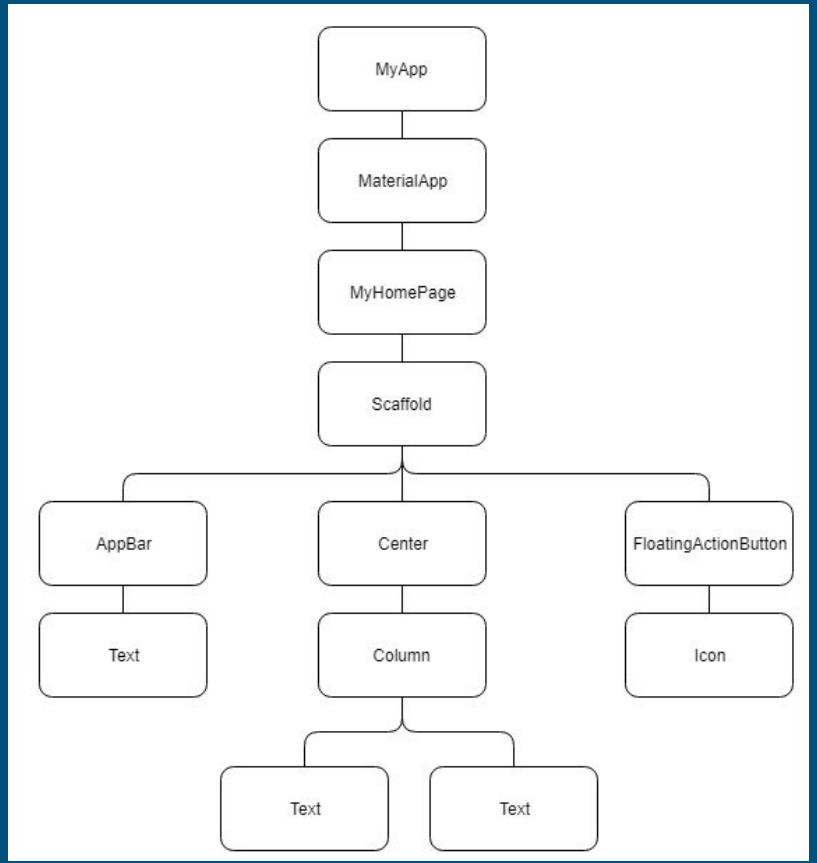
Les bases

Notion d’arborescence Widgets

Ils sont organisés sous forme d’arbre (tree)

Un Widget parent contient d’autres Widgets qui sont appelés Widgets enfants

Les bases



Les bases

Notion de Context

Un **context** est une référence à l'emplacement d'un Widget dans l'arborescence de tous les Widgets qui sont construits

Un context n'est lié qu'à un seul Widget

Si un widget 'A' contient des widgets enfants, le context de widget 'A' deviendra le context parent de tous les enfants directs

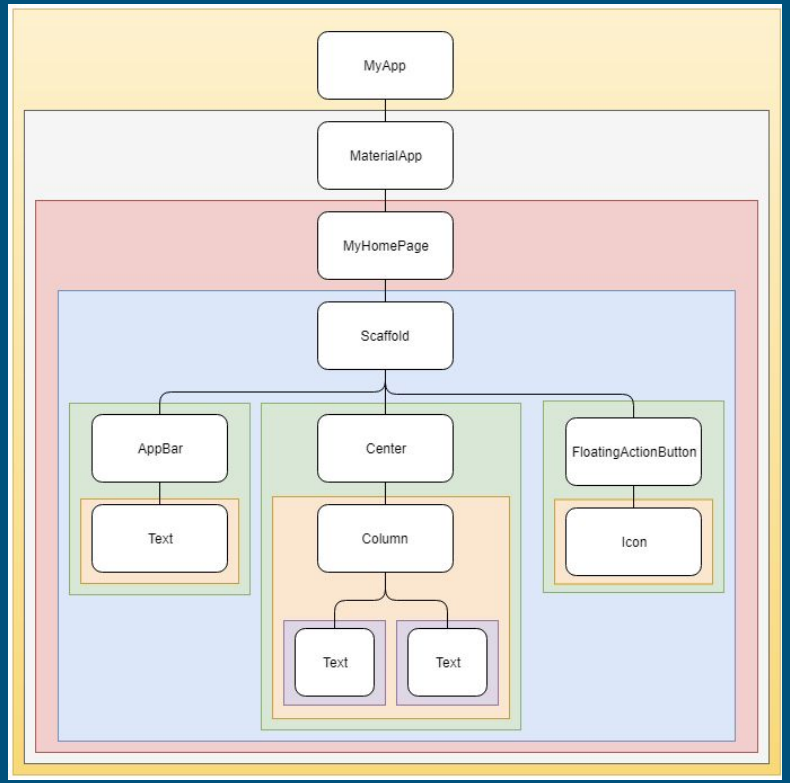
Les bases

Notion de Context

Les contextes sont liés et eux aussi composent une arborescence de contextes (relation parent-enfants)

Pour illustrer cette notion, on peut reprendre l'arborescence précédente, chaque couleur représente un context

Les bases



Les bases

Context Visibility

Quelque chose n’est visible que dans son propre contexte ou dans celui de ses parents

À partir d’un context enfant, on peut donc retrouver un widget ancêtre

Les bases

Context Visibility

Exemple:

Scaffold > Center > Column > Text:
context.ancestorWidgetOfExactType(Scaffold)

On récupère le premier Scaffold en remontant dans l’arborescence à partir du context du Text

Les bases

Types de Widgets

Il existe 2 types de widgets

- Stateless Widget
- Stateful Widget

Les bases

Stateless Widget

Certains composants visuels ne dépendent pas d'autre chose que de leurs propres informations de configuration (fournies à la construction)

Ces Widgets ne varient pas une fois créés

Exemple:

Text, Row, Column, Container

Les bases

Stateless Widget - Cycle de vie

- initialisation (initState)
- construction via la méthode build

Les bases

Stateless Widget

```
class MyStatelessWidget extends StatelessWidget {

        MyStatelessWidget({
                Key key,
                this.parameter,
        }): super(key:key);

        final parameter;

        @override
        Widget build(BuildContext context){
                return new ...
        }
}
```

Les bases

Stateful Widget

Certains widgets vont gérer certaines données internes qui changeront pendant la durée de vie du Widget, ces données deviennent donc dynamiques

Les données contenues et gérées par ce Widget et qui peuvent varier pendant la durée de vie du Widget s'appellent un **State**

Exemple,

Liste de cases à cocher, bouton activer/désactiver en fonction d'une condition

Les bases

Statefull Widget - Cycle de vie

- initialisation (initState)
- construction via la méthode build
- suppression (dispose)

Les bases

Pour connaître les composants de base et la façon de les utiliser nous allons effectuer les code lab ci-dessous :

https://docs.flutter.dev/get-started/codelab-web

https://codelabs.developers.google.com/codelabs/flutter/index.html

Navigation

Vous pouvez utiliser les méthodes de la classe Navigator pour naviguer dans votre application

- push
- pop
- pushReplacement
- pushAndRemoveUntil

Codelab => https://flutter.io/cookbook/navigation/navigation-basics/

Navigation

Vous pouvez aussi donner des noms à vos routes afin de simplifier la navigation

Il suffit d'associer un écran à un nom et d'utiliser ce nom lors du changement d'écran. Cette association est réalisée via l'attribut **routes** de la classe **MaterialApp**

- pushNamed
- pop
- pushReplacementNamed
- pushNamedAndRemoveUntil

Codelab => https://flutter.io/cookbook/networking/named-routes/

Drawer

Vous pouvez ajouter facilement un menu à votre application grâce à la classe **Drawer**

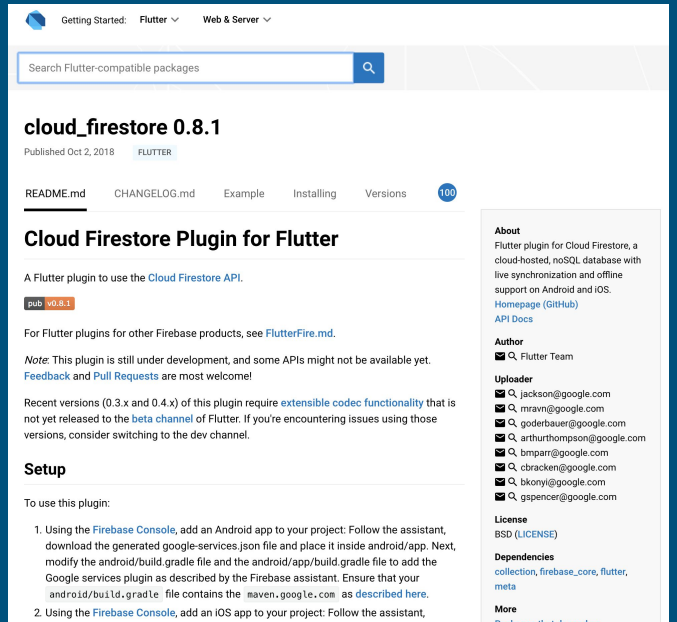
Il suffit d'utiliser un **Scaffold**, qui contient un attribut **drawer** qui permet de mettre en œuvre facilement notre propre implémentation de Drawer.

Pour cela on instancie un ListView au niveau de l'attribut child de notre Drawer

Codelab => https://docs.flutter.dev/cookbook/design/drawer

Librairie externe

Vous pouvez utiliser le portail https://pub.dev/ pour rechercher des librairies externes afin d'ajouter des fonctionnalités à votre application



Librairie externe

Il suffit d'ajouter le plugin dans le fichier **pubspec.yaml** situé à la racine du projet

```yaml
dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^0.1.0
  # Redux
  redux: ^3.0.0
  flutter_redux: ^0.5.2
  redux_logging: ^0.3.0
  redux_thunk: ^0.2.0
  redux_persist_flutter: ^0.6.0-rc.1
  redux_persist: ^0.7.0-rc.2
  redux_epics: ^0.10.2
  rxdart: ^0.18.1

  intro_views_flutter: ^2.2.4
  firebase_auth: ^0.5.20
  #google_sign_in: ^3.2.1
  cloud_firestore: ^0.8.1
  fluro: ^1.3.4
  intl: ^0.15.7
  uuid: ^1.0.3
  flutter_udid: ^0.0.3
```

Librairie externe

Pour mettre en pratique l'utilisation de

- Librairies externes
- Firebase
- Introduction au gestionnaire de State (Provider)

Nous allons faire le codelab :

https://firebase.google.com/codelabs/firebase-get-to-know-flutter

TP de groupe

Objectif:

Utiliser les différents outils de Firebase (ou Supabase / AppWrite) pour construire une application complète

Une entreprise de vente de produits en porte à porte souhaite avoir une application permettant de suivre l'évolution de son chiffre d'affaire en temps réel

Elle est composée de 10 commerciaux et 4 techniciens

TP de groupe

Lors d'un rendez-vous lorsqu'un commercial réalise une vente, on stocke les informations en base de données *//l'ensemble des employés reçoit une notification sur son téléphone*

Un écran liste les ventes réalisées et est actualisé en temps réel *//Le classement des commerciaux est visible sur une TV dans la salle de réunion de l'entreprise*

Les ventes sont saisies sur téléphone à la sortie du rendez-vous

TP de groupe

Comment faire? => Utilisation de Firebase pour gérer:

- L'authentification (https://firebase.google.com/docs/auth/flutter/start)
- Le stockage des données dans FireCloud (https://firebase.google.com/docs/firestore/quickstart#dart_1)
- // Le stockage des images (Firebase Storage)
- // Les notifications
- // L'hébergement web

TP de groupe

Chaque vente passe par différents statut (vendu, visite technique validée, financement validé / annulation)

On peut accéder au détail des statistiques de chaque commercial / prospecteur

TP de groupe

Points Bonus:
Nouvelle fonctionnalité non listée dans le sujet (notification de rappel de rendez-vous…)

Conseil de départ:
Commencez par définir en équipe le modèle de données dans Firestore avec quelques exemples concrets (liste des médecins, rendez-vous…). Allez-vous utiliser des librairies externes? Si oui, lesquelles et dans quel objectif?. Découpez le travail en “petites tâches” et répartissez les entre vous.

Méthode de rendu:
Les sources des projets doivent être postées sur github, une personne de l’équipe m'envoie un mail (pierre.deruel@envoxoo.com) avec les liens vers les projets, le nom et prénom des personnes de son équipe et éventuellement la documentation et/ou les commentaires associés aux projets.