

Chapitre 1: Introduction à Flutter

▼ Qu'est-ce que Flutter ?

Flutter est un framework open-source de développement d'applications mobiles créé par Google en 2017.

Flutter a été déclaré stable et est sorti de bêta en juin 2018 et sa version 3.7 est sortie en 2023, il s'agit donc du moment parfait pour l'apprendre !

Il utilise **Dart** qui est un langage créé par Google qui peut être compilé en code bas niveau ou transpilé en **JavaScript**

Il est utilisé par Google pour développer la plupart de ses applications mobiles externes comme par exemple celle pour **Google Ads**, ou internes comme **Google Greentea** (permettant de gérer ses relations clients).

Il est également utilisé en production par Alibaba pour des applications avec plus de 50 millions de téléchargements

Il est enfin utilisé pour développer des applications sur le nouveau système d'exploitation de Google dénommé **Google Fuchsia**.

Fuchsia

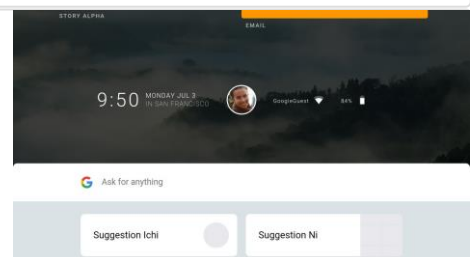
Fuchsia is a new open source operating system

🔗 <https://fuchsia.dev/>

Google Fuchsia

Fuchsia est un système d'exploitation développé par Google. Contrairement aux précédents systèmes d'exploitation développés par Google tels que Google


🌐 https://fr.wikipedia.org/wiki/Google_Fuchsia



Vous pouvez voir ici des centaines d'exemples d'application réalisées avec Flutter !

Flutter Apps | It's All Widgets!

An open list of example apps made with Flutter includes many open source samples.

 <https://itsallwidgets.com/>



Avantages de Flutter

Nous allons commencer par un bref récapitulatif des avantages à utiliser Flutter puis nous rentrerons plus dans les détails.

- 1 ? **Pouvoir créer des applications hybrides réactives sans utilisation de pont JavaScript / Plateforme native** dégradant les performances.
- 2 ? **Code pré-compilé ?AOT?** en code natif bas niveau augmentant les performances.
- 3 ? **Widgets** et **Layout** totalement personnalisables sans perte de performance.
- 4 ? Une très importante librairie de **widgets** de qualité Google.
- 5 ? Utilisation intensive par les équipes de Google, ce qui assure sa maintenabilité, sa qualité et des mises à jour très proches de celles d'Android et d'iOS.
- 6 ? Le prochain **OS** de Google appelé **Fuchsia** utilise **Flutter**, ce qui assure également l'utilisation d'un framework **future proof**(pérennité) est le processus d'anticipation du futur et développer des méthodes de minimiser les effets des chocs et des contraintes des événements futurs.

En résumé, **Flutter** est le framework de développement hybride le plus moderne, le plus performant, avec une équipe de développement Google dédiée et un accès privilégié aux futures avancées des plateformes mobiles (**Android**, **Fuchsia** étant développés par Google).

Historique des solutions pour le développement d'applications mobiles

Nous allons voir à travers un bref historique pourquoi **Flutter** est l'approche la plus moderne et la plus innovante en matière de développement mobile hybride.

- **Les SDK pour les plateformes**

SDK est l'acronyme anglais de « **Software Development Kit** » ou « **Kit de développement** ». Il regroupe un ensemble d'outils d'aide à la programmation d'applications mobiles.

Les **SDK** de développement mobile ont environ dix ans avec le **SDK iOS** qui est sorti en 2008 et le **SDK Android** de Google qui est sorti en 2009.

Les langages utilisés au départ étaient uniquement Objective-C pour iOS et Java pour Android.

Le fonctionnement d'une application mobile est d'interagir avec la plateforme mobile pour créer des **widgets** ou accéder à des **services natifs comme la caméra**.

Pour ce faire, elle utilise un langage natif pour la plateforme, et il faut donc créer une application différente pour chaque plateforme.

Il existe aujourd'hui une multitude d'**OS** et de **SDK** mais Android représente environ **75%** du **marché**, suivi par iOS à **22%** environ.

Pour vous donner un ordre d'idée ou **Windows Phone** **Samsung** représente chacun environ **0,2%**.

- **Les vues Webs**

Les premiers **frameworks** permettant de construire des applications multiplateformes utilisent le **JavaScript** et les **Webviews**.

Ces **frameworks** sont par exemple **Ionic**, **PhoneGap** et **Cordova**.

Ils créent en fait des vues webs en HTML qui sont rendus par le mobile et utilisent des API faisant le pont avec les services bas niveau du téléphone (caméra, audio, position, bluetooth etc).

Elle permet une réutilisation d'une partie du code **Angular / React / Vue** avec **Ionic 4** mais par design il ne sera jamais possible d'approcher des performances natives.

- **Les vues réactives**

Avec l'apparition des **single page applications** et de l'utilisation de la programmation réactive pour le Web, des nouveaux frameworks ont fait leur apparition utilisant ce paradigme, par exemple **React Native**.

Le problème est que ces applications utilisaient du **JavaScript** qui communiquent avec des **API** faisant le pont vers les services bas niveau et la création de **widgets**, ce qui dégradent les performances de l'app.

Flutter fait partie de cette catégorie de **frameworks** et utilisent des vues réactives.

Mais la première différence est que le code `Dart` utilisé est compilé Ahead Of Time (AOT) en langage bas niveau.

Les performances sont ainsi largement augmentées : aussi bien pour le démarrage, que pour l'accès aux services natifs ou à la génération des vues.

Flutter utilise en outre une approche très innovante pour l'utilisation des widgets.

Que sont les widgets ?

Un widget est un élément qui contrôle la vue et est géré par l'application.

Il existe pour chaque plateforme une librairie de widgets, et pour les frameworks utilisant des vues Web des widgets utilisant le DOM.

NB Le DOM (Document Object Model) est une interface pour vos pages web. C'est une API permettant aux programmes de lire et de manipuler le contenu de la page, sa structure et ses styles.

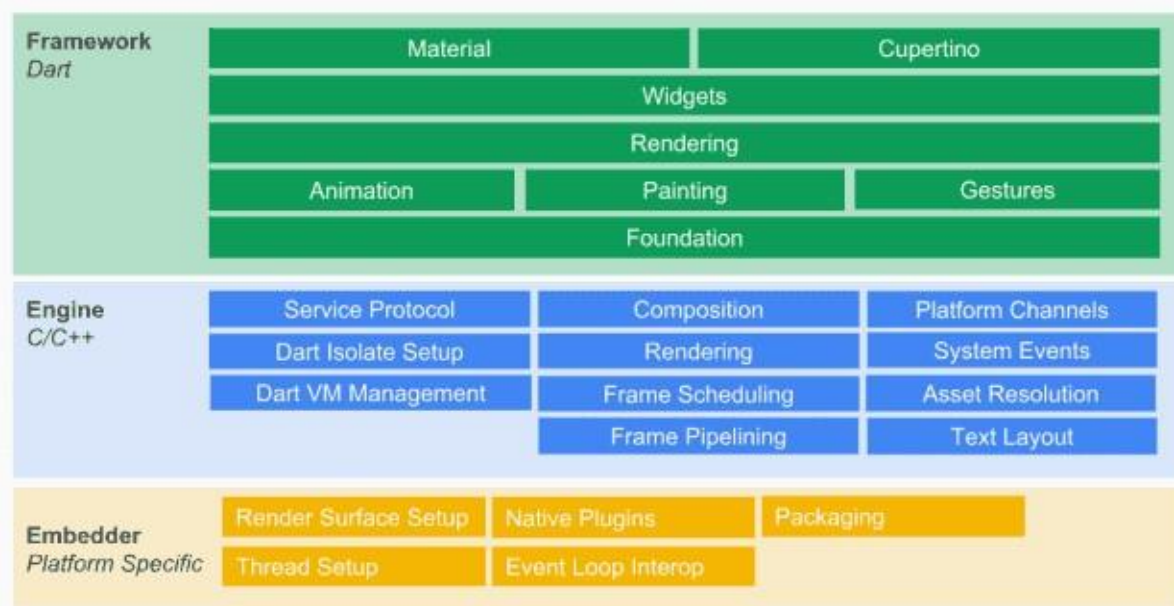
Flutter prend une approche différente en proposant sa propre librairie de widgets de très haute qualité et totalement personnalisable.

La création des vues et le rendu est donc du côté Flutter et non pas du côté plateforme (que ce soit pour les vues Web ou l'utilisation de widgets natifs). Il ne reste que la peinture des vues (phases paint) dans les canvas.

NB Canvas est un widget permettant d'afficher des éléments graphiques tels que des lignes ou du texte etc.

Même pour la phase de paint, Flutter ne met à jour que les widgets en ayant besoin ce qui améliore encore plus les performances

Flutter n'accède donc qu'à trois ensembles bas niveau : les events, les canvas (utilisés pour le rendu des widgets) et les services. Voici une vue d'ensemble de l'architecture de Flutter :



Bref comparatif avec React Native

J'ai choisi de vous présenter un comparatif avec **React Native**, car il s'agit de la solution qui vient souvent à l'esprit des développeurs souhaitant utiliser un framework permettant de développer des applications hybrides.

React Native a de moins bonnes performances principalement car il n'est pas compilé en code bas niveau mais utilise un pont vers la plateforme mobile. Cette architecture dégrade les performances.

Les composants disponibles sur **React Native** sont **basiques**, donc si vous voulez faire des choses un peu plus **complexe** il vous faudra beaucoup de développement et vous devrez utiliser la plupart du temps des composants différents entre **iOS** et **Android**. Vous utiliserez souvent des **librairies tierces qui ne sont pas supportés par Facebook ni aucune grande entreprise**, ce qui implique qu'elles peuvent être **abandonnées** du jour au lendemain.

React Native est limité par l'UI de la plateforme (soit iOS, soit Android) car il utilise le rendu de la plateforme. Ce n'est pas le cas avec **Flutter**, comme nous le verrons.

Les **toolkits** natifs sont fragmentés en fonction des versions **d'Android** et **d'iOS** car de nouvelles UI sont ajoutées, comme par exemple de nouvelles animations, mais elles ne sont pas **compatibles** avec les **anciennes versions**. Il est donc difficile de créer des UI fonctionnant sur toutes les versions, même récentes, avec React Native ou des applications natives. Avec **Flutter**, le rendu est effectué côté Flutter et non côté plateforme, tous ces problèmes de compatibilité n'existent donc pas.

React Native n'a pas de **librairie officielle de tests d'intégration** et **d'UI** contrairement à **Flutter**.

React Native n'a pas non plus de librairie officielle pour **l'intégration et le déploiement continus** (continuous integration / continuous delivery), vous ne pouvez donc pas automatiser vos build et envois aux stores. Si vous voulez le faire, vous devrez utiliser des **services tiers payants** et non supportés par Facebook. **Flutter** a une **solution de CD/CI supportée par Google**.

Flutter est plus moderne, beaucoup plus performant et plus **future proof** que **React Native**. Si vous ne connaissez aucun des deux nous vous recommandons très fortement **Flutter** !

▼ Qu'est-ce que Dart?

Dart est un langage développé par Google initialement pour remplacer **JavaScript**. La première version est sortie en 2011.

Aujourd'hui l'objectif de **Dart** est de pouvoir être facilement transpilé en **JavaScript** pour le Web et compilé en langage bas niveau pour les applications natives.

La version 2 du langage est sortie en août 2018 et apporte de grandes avancées que nous traiterons.

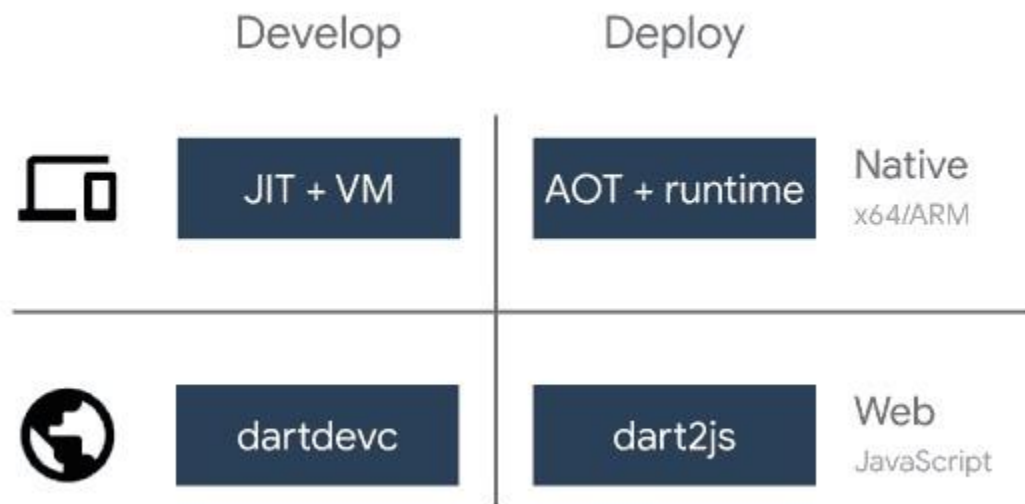
Dart est un des langages les plus utilisés chez Google. Il est utilisé pour les applications les plus importantes en Web : Google Ads, Google Shopping et des dizaines d'applications internes.

Il est de plus en plus utilisé, et par exemple **Sass**, le préprocesseur CSS, a été réécrit récemment en **Dart**.

Avec Dart vous êtes certains de pouvoir développer des applications scalables. Google l'utilise en production pour des dizaine de millions de ligne de codes.

▼ Pourquoi utiliser Dart ?

Dart possède plusieurs compilateurs pour le Web et le mobile :



Compilation en langage bas niveau pour le mobile

Dart est un langage qui peut être compilé en **AOT** en code natif ce qui lui confère des performances excellentes en production.

Il dispose également d'un compilateur **JIT** pour le développement afin de permettre une expérience de développement optimale avec le hot reload (voir plus bas).

Un langage facile pour raisonner

Dart est un langage **single threaded** comme **JavaScript** lui permettant d'éviter plusieurs problèmes dont les race conditions comme dans les langages supportant plusieurs **threads** d'exécution concomitants : ce qui inclut **Java**, **Kotlin**, **Objective-C** et **Swift**.

NB? "single-threaded" signifie qu'il n'exécute qu'une seule tâche à la fois. Cela veut dire que le code Dart est exécuté de manière séquentielle, ligne par ligne, sans la possibilité d'exécuter simultanément plusieurs.

Les race conditions se produisent lorsque plusieurs threads tentent d'accéder et de modifier les mêmes données simultanément, ce qui peut entraîner des comportements indéterminés et des bugs difficiles à reproduire et à résoudre.

En étant **single-threaded**, Dart simplifie la gestion des accès concurrents aux données, car il n'y a pas de besoin de synchroniser les threads ou de se préoccuper des problèmes de concurrence. Cela facilite le développement de code robuste et prévisible, en réduisant la complexité liée à la gestion des threads et des verrous.

Cependant, il est important de noter que bien que Dart soit single-threaded, il offre toujours la possibilité de travailler de **manière asynchrone** grâce aux **async/await**, les **Futures** et les **Streams**, que nous étudierons, afin de gérer l'asynchrone de manière optimale sur un seul **thread**. Cela permet d'exécuter des tâches en arrière-plan de manière non bloquante, ce qui est particulièrement utile pour les opérations réseau, les opérations d'E/S ou d'autres tâches intensives en termes de temps.

Un langage facile à apprendre pour tous les développeurs Web

Dart est facile à apprendre que ce soit pour des développeurs venant de langages dynamiques (comme **JavaScript**) ou statiques (comme **C ou Java**).

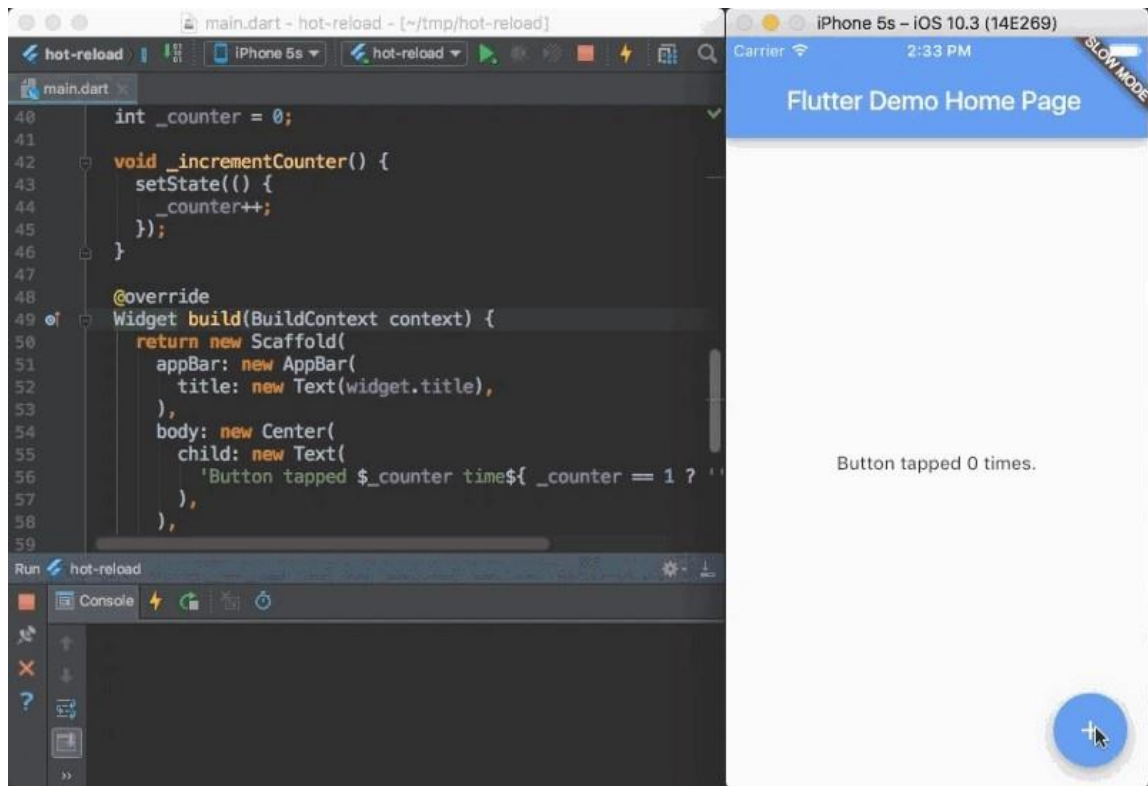
Dart et Flutter

Nous allons maintenant voir quels sont les avantages d'utiliser **Dart** avec **Flutter**.

- **Le rechargement rapide(Hot reload)**

Une fonctionnalité géniale en développement est le **hot reload** !

Grâce à **Dart**, et à son compilateur **JIT**, **Flutter** permet de recharger que le code qui a changé pendant que l'application tourne, ce qui permet d'avoir la nouvelle version en moins d'une seconde, **le tout en conservant son état** !



- **Utilisation mémoire optimale pour l'affichage d'animations parfaitement fluides**

Dart est optimal pour effectuer des animations et des transitions qui sont à **60 fps (frames per second)**. En effet, il dispose d'un **garbage collector** et d'une **allocation mémoire** pour les objets permettant de meilleures performances.

NB? 60 fps signifie "60 frames per second" (60 images par seconde). Dans le contexte des applications et des jeux, le nombre d'images par seconde se réfère au taux de rafraîchissement de l'affichage.

Lorsqu'une application ou un jeu fonctionne à 60 fps, cela signifie que l'affichage est mis à jour 60 fois par seconde. Chaque image affichée est appelée une "frame". Plus le nombre de frames par seconde est élevé, plus l'animation ou l'interaction semble fluide et réactive à l'œil humain.

Un taux de rafraîchissement plus élevé, comme 60 fps, donne une sensation de fluidité et de réactivité accrue à l'utilisateur. Cela est particulièrement important dans les applications graphiques, les jeux et les interfaces utilisateur où des mouvements rapides et fluides sont nécessaires pour offrir une expérience agréable.

Ce design du langage est parfait pour les applications mobiles qui requièrent un sentiment de fluidité parfaite lors de l'utilisation.

- **Un seul langage, même pour le Layout**

Avec **Flutter**, les **layouts** sont également écrits en **Dart**, par exemple :

```
Center(child:
  Column(children: [
    Text('Bonjour cher apprenant !'),
    Icon(Icons.star, color: Colors.blue),
  ])
)
```

Il n'y a pas besoin de langage spécifique pour le Layout comme **JSX** ou **XML**.

NB? Un layout dans Flutter est construit à l'aide d'une hiérarchie de widgets, où chaque widget est responsable de la disposition de ses enfants. Les widgets parent contrôlent la façon dont les widgets enfants sont positionnés les uns par rapport aux autres.

Exemple:

Container, Row, Column, Stack, Expanded, GridView

▼ Installation de l'environnement

NB? Il est obligatoire d'installer Android Studio pour pouvoir utiliser le SDK Android et l'émulateur..

L'émulateur ios n'est disponible que sur mac.

▼ Environnement Windows

Install

Install Flutter and get started. Downloads available for Windows, macOS, Linux, and ChromeOS operating systems.

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



?? Installation de Git

Sur **Windows**, téléchargez et installez **Git** en utilisant l'exécutable officiel que vous trouverez ici

Git ? Downloading Package

 <https://git-scm.com/download/win>

?? Installation du SDK Flutter

Pour installer le **SDK Flutter** stable, il suffit d'aller ici.

Windows install

How to install on Windows

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



Téléchargez le **SDK** en cliquant sur le bouton bleu.

Ensuite il faut extraire le fichier téléchargé mais pas n'importe où !
Il faut l'extraire par exemple dans : **C:\src\flutter**.

?? Modification du **PATH**

Dans la barre de recherche **Windows** tapez "environnement" ou "env" puis sélectionnez **Modifier Variables d'environnement** puis **Variables d'environnement**, cherchez la variable **Path** et cliquez sur modifier.

Cliquez sur "Parcourir..." et allez dans le dossier où vous avez extrait Flutter ou copiez collez directement le chemin.

?? Vérification de l'installation

Vous pouvez ensuite vérifier ce qu'il nous reste à installer avec :


```
flutter doctor
```

Il faut installer tout ce qui manque.

?? Installation d'**Android Studio**

Pour installer **Android Studio**, il suffit d'aller ici.

Download Android Studio & App Tools  Android Developers

Android Studio provides app builders with an integrated development environment  optimized for Android apps. Download Android Studio today. <https://developer.android.com/studio>



Vous devez ensuite l'extraire et le mettre à l'endroit que vous voulez, par exemple au même endroit que flutter.

?? Installation des paquets dans **Android Studio**

Dans **Android Studio** laissez tout par défaut et faites **next** plusieurs fois puis **finish**.

Cela va télécharger les premiers paquets nécessaires. Redémarrez ensuite **Android Studio**.

Allez dans **Tools** → **SDK Manager** → **SDK Tools**.

Cochez **Android SDK Command-line tools**.

Cliquez sur **Apply**.

?? Accepter les licences d'utilisation

Ensuite ouvrez un nouveau terminal et faites :

```
flutter doctor --android-licenses
```

Refaites une dernière fois :

```
flutter doctor
```

Vous devez avoir tout passé en vert et avoir le message **No issues found!**.

Si ce n'est pas bon reprenez ce qu'il manque.

?? Installation de VS Code

C'est un éditeur de code créé par **Microsoft** et qui est gratuit. Il est très complet et très simple à prendre en main.

Pour installer **VS Code**, c'est ici.

```
https://code.visualstudio.com/download
```

?? Installation des extensions VS Code.

Dans **VS Code** allez dans l'onglet extension à gauche.

Recherchez **flutter** et installez la première.

??? Création d'un mobile virtuel

Allez dans **Android Studio** → **Tools** → **Android** → **AVD Manager** puis cliquez sur **Create Virtual Device**.

Choisissez par exemple **Pixel 6** puis faites **Next**.

Installez ensuite une version **d'Android** puis faites **Next**.

Cliquez enfin sur **Finish**.

Enfin cliquez sur la flèche verte **Run** pour lancer l'émulateur.

Vous pouvez cliquer sur la roue de paramètres à droite vers le bas puis cliquer sur **View Mode** puis **Float**.

▼ Environnement GNU/Linux

Nous allons maintenant mettre en place l'environnement de développement Flutter !

?? Installation de VS Code

Nous recommandons VS Code. C'est un éditeur de code créé par Microsoft et qui est gratuit. Il est très complet et très simple à prendre en main.

Pour installer VS Code, c'est ici

```
https://code.visualstudio.com/download
```

Vous pouvez également l'installer avec snap.

```
sudo snap install --classic code
```

?? Installation du SDK Flutter

Pour installer le SDK Flutter stable, il suffit d'aller ici.

Choisissez ensuite Linux.

Entrez simplement la commande suivante dans votre terminal :

```
sudo snap install --classic flutter
```

?? Vérification de l'installation

Vous pouvez ensuite vérifier ce qu'il nous reste à installer avec :

```
flutter doctor
```

Vous obtiendrez quelque chose comme cela :

```
Downloading https://storage.googleapis.com/flutter_infra_release/releases/stable/linux/flutter_linux_3.3.9-stable.tar.xz
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left     Speed
100 503M 100 503M    0     0  40.0M      0  0:00:12  0:00:12 --:--:-- 41.7M
Flutter initialized
Flutter 3.3.9 • channel stable • https://github.com/flutter/flutter.git
Framework • revision b8f7f1f986 (4 days ago) • 2022-11-23 06:43:51 +0900
Engine • revision 8f2221fbef
Tools • Dart 2.18.5 • DevTools 2.15.0
Flutter assets will be downloaded from https://storage.googleapis.com. Make sure you trust this source!
Running "flutter pub get" in flutter_tools... 9.4s
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (channel stable, 3.3.9, on Ubuntu 22.04.1 LTS 5.15.0-53-generic, locale en_US.UTF-8)
[✓] Android toolchain - develop for Android devices
[✗] Unable to locate Android SDK.
     Install Android Studio from: https://developer.android.com/studio/index.html
     On first launch it will assist you in installing the Android SDK components.
     (or visit https://flutter.dev/docs/get-started/install/linux#android-setup for detailed instructions).
     If the Android SDK has been installed to a custom location, please use
     'flutter config --android-sdk' to update to that location.
[✓] Chrome - develop for the web
[✓] Linux toolchain - develop for Linux desktop
[!] Android Studio
     android-studio-dir = /snap/android-studio/current/android-studio
     Android Studio not found at /snap/android-studio/current/android-studio
[✓] VS Code
[✓] Connected device (2 available)
[✓] HTTP Host Availability

! Doctor found issues in 2 categories.
```

Pas de panique ! Nous allons compléter les éléments manquants.

?? Installation d'Android Studio

Pour installer Android Studio, il d'ouvrir un terminal et de faire :

```
sudo snap install --classic android-studio
```

?? Créer un raccourci

Pour créer un raccourci allez dans Android Studio puis Tools puis Create Desktop Entry.

Vous pouvez ensuite le retrouver facilement et l'ajouter en favori.

?? Installation des paquets dans Android Studio

Dans Android Studio laissez tout par défaut et faites **next** plusieurs fois puis **finish**.

Cela va télécharger les premiers paquets nécessaires.

Allez dans **File ? Settings ? Plugins**.

Dans Tools cliquez sur SDK Manager puis cliquez sur l'onglet SDK Tools, cochez Android SDK Command-line Tools et cliquez sur OK.

Ensuite cliquez sur restart.

?? Création d'un mobile virtuel

Une fois relancé allez dans Android Studio ? Tools ? Device Manager puis cliquez sur Create device.

Choisissez par exemple Pixel 6 puis faites Next.

Installez ensuite une version d'Android par exemple Tiramisu puis faites Finish.

Enfin cliquez sur la flèche verte Run pour lancer l'émulateur.

Vous pouvez cliquer sur la roue de paramètres à droite vers le bas puis cliquer sur View Mode puis window.

?? Accepter les licences d'utilisation

Ensuite ouvrez un autre terminal et faites :

```
flutter doctor --android-licenses
```

Faites y à tout pour accepter les licences d'utilisation d'Android.

Vous pouvez enfin faire :

```
flutter doctor
```

A ce stade normalement tout est vert.

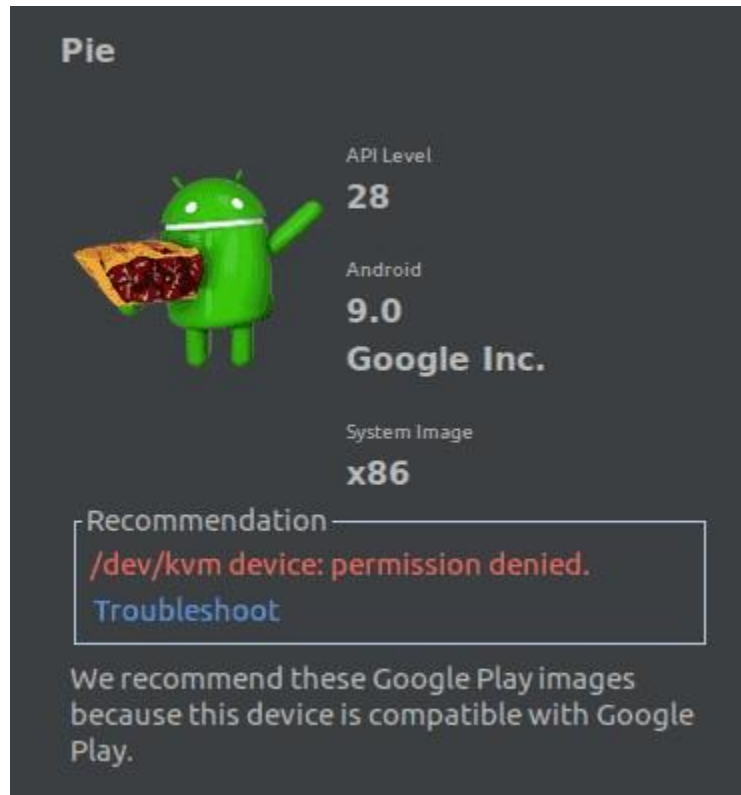
Si il reste un paquet à installer sur votre système suivez les instructions fournies.

?? Utilisation de VS Code

Allez dans **Extensions** puis installez **flutter**.

??? Problème éventuel de permission

Si vous utilisez Ubuntu vous pourriez rencontrer le problème suivant :



Pour le résoudre faites :

```
sudo apt install qemu-kvm
sudo adduser UTILISATEUR kvm
sudo chown UTILISATEUR /dev/kvm
```

Remplacez utilisateur par votre **username** qui est indiqué dans votre terminal avant le **@**, par exemple babs au-dessus.

Relancez ensuite **Android Studio**.

▼ Environnement macOS

🔗 Installation de Visual Studio Code

Pour installer Visual Studio Code, c'est ici.

<https://code.visualstudio.com/>

Téléchargez simplement l'installateur macOS.

?? Installation des extensions VS Code

Dans **VS Code** allez dans l'onglet extension à gauche.

Recherchez **flutter** et installez la première.

?? Installation du SDK Flutter

Pour installer le **SDK Flutter** stable, il suffit d'aller ici.

macOS install

How to install on macOS.



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



Téléchargez le **SDK** en cliquant sur le bouton bleu.

Déplacez le dans votre dossier utilisateur, par exemple **babs**.

?? Mise à jour du PATH

Il faut indiquer au **shell** où trouver le programme **flutter** lorsque l'on tape **flutter** dans le terminal. Pour ce faire, il faut mettre à jour le **PATH** qui contient les chemins vers les exécutables.

Si vous avez le **shell zsh** :

```
cd
nano .zshrc
```

Dans le fichier mettez simplement :

```
export PATH="$PATH:$HOME/flutter/bin"
```

Sauvegardez le fichier et quittez le terminal.

?? Vérification de l'installation

Vous pouvez ensuite vérifier ce qu'il nous reste à installer avec :

```
flutter doctor
```

Il faut installer tout ce qui manque.

?? Installation de Xcode

Si vous n'avez pas XCode installez le depuis ici.

Xcode

Xcode includes everything developers need to create great applications for Mac, iPhone, iPad, Apple TV, and Apple Watch. Xcode provides



<https://apps.apple.com/us/app/xcode/id497799835>



Une fois l'installation terminée, ouvrez un terminal et faites :

```
sudo xcode-select --switch /Applications/Xcode.app/Contents/Developer
sudo xcodebuild -runFirstLaunch
sudo xcodebuild -license
```

Lorsque cela vous est demandé, acceptez les licences en tapant agree puis entrée.

?? Configurez l'émulateur

Ouvrez un terminal et tapez :

```
open -a Simulator
```

?? Installation de cocoapods

Ouvrez un terminal et faites :

```
sudo gem install cocoapods
```

Si cela ne fonctionne pas, utilisez brew :

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/brew/HEAD/
```


Puis :

```
brew install cocoapods
```

?? Installation d'Android Studio

Pour installer Android Studio, il suffit d'aller ici.

Download Android Studio & App Tools  Android Developers

Android Studio provides app builders with an integrated development environment  IDE optimized for Android apps. Download Android Studio today. <https://developer.android.com/studio>



??? Configuration d'Android Studio Lancez

Android Studio.

Utilisez l'installation standard et acceptez toutes les licences dans les onglets de gauche.

Allez sur le menu **Tools** puis **SDK Manager** puis **Android SDK** puis **SDK Tools** et cochez **Android SDK Command-line Tools**.

Faites ensuite **Apply** puis **Ok**.

Ouvrez un terminal pour accepter les licences Android :

```
flutter doctor --android-licenses
```

Acceptez les licences avec y puis entrée.

??? Vérification des installations

Vérifiez l'installation en faisant :

```
flutter doctor
```

??? Test de lancement d'une application

Retourner sur votre HOME 

```
cd
```

Créez une application flutter :

```
flutter create my_app
```

Allez dans le dossier :

```
cd my_app
```

Essayez de lancer l'application :

```
flutter run
```

▼ Ma Première application

?? **Création d'un nouveau projet Flutter** Lancez

Android Studio ou VS Code.

- Créer un projet avec le CLI

Nous allons utiliser le CLI flutter pour créer un projet.

Ouvrez un terminal et faites :

```
flutter create my_first_app
```

```
cd create my_first_app
```

```
flutter run my_first_app
```

- Créer un projet sur Android Studio

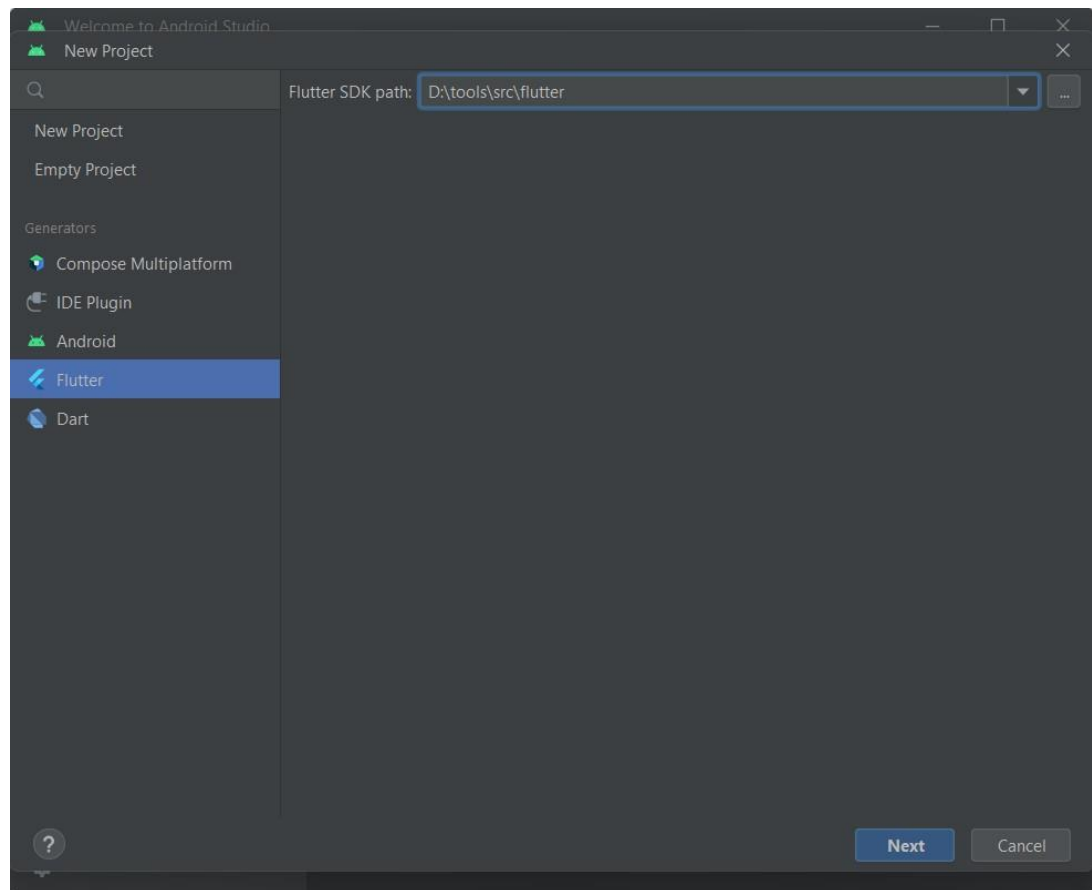
Sur Android Studio il suffit de faire Start a new Flutter Project sur l'écran d'accueil.

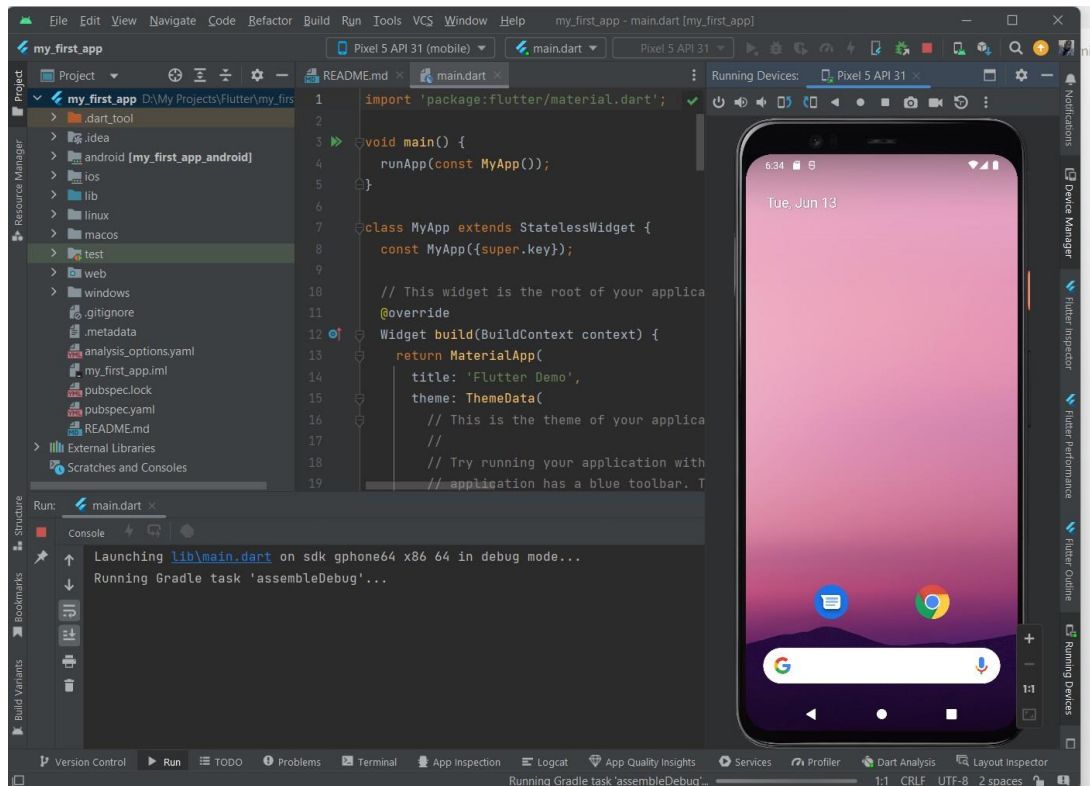
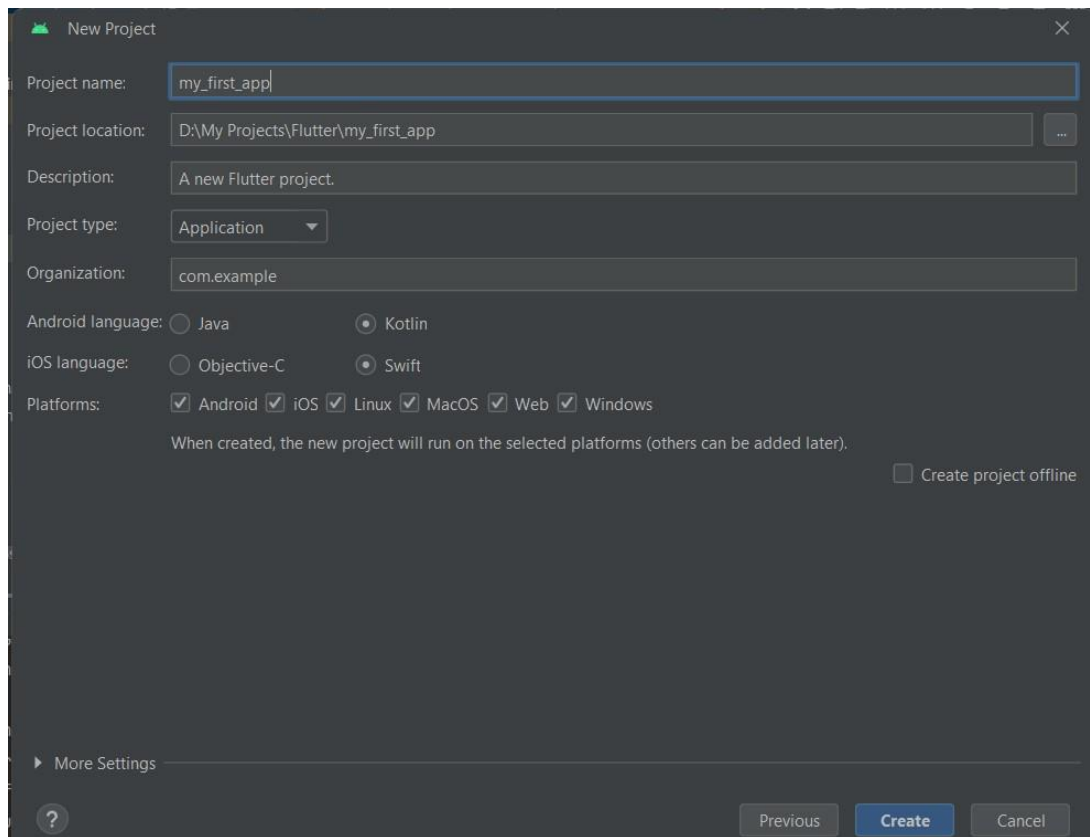
Sélectionnez ensuite Flutter Application puis faites Next.

Vous pouvez ensuite choisir le nom de l'application. Faites ensuite Next.

Pour notre application d'exemple, faites simplement Finish sur l'écran suivant.

Les fichiers seront automatiquement créés.





?? Etude des fichiers

Nous allons faire un premier tour des fichiers générés par le CLI flutter.

- Les fichiers **pubspec.yaml** et **pubspec.lock**

En utilisant **Dart**, nous adoptons un écosystème différent de celui utilisé pour le **JavaScript**.

En **JavaScript** vous êtes habitué aux fichiers **package.json** et **package-lock.json** pour la gestion des dépendances de votre projet.


En **Dart**, nous n'utilisons pas le **package manager** de **Node.js**, **npm**.

Nous utilisons un autre gestionnaires de paquets appelé le **pub package manager**.

Le site de référence permettant de rechercher des packages est

Dart packages

Pub is the package manager for the Dart programming language, containing reusable libraries & packages for Flutter and general Dart

 <https://pub.dev/>



Le fonctionnement est très similaire de **npm**.

Vous avez un fichier **pubspec.yaml** qui contient la liste des dépendances de votre projet au format YAML.

Nous verrons dans les chapitres suivant comment installer, mettre à jour et importer des dépendances.

Le fichier **pubspec.lock** permet de lister les versions de chaque dépendance de votre projet pour obtenir exactement le même arbre de dépendances entre deux environnements (comme pour le **package-lock.json**).

- Le fichier **my_first_app.iml**

Il s'agit d'un fichier de configuration pour les éditeurs. Vous n'aurez pas à le modifier.

- Le fichier **.packages**

Il s'agit d'un fichier caché qui contient les chemins vers les dépendances de votre application. Vous n'aurez jamais à le modifier.

- Le fichier **.metadata**

Il s'agit d'un fichier caché qui contient des informations pour l'éditeur de code relative au projet Flutter.

- Le fichier **.gitignore**

Il s'agit d'un fichier caché qui est utilisé par **Git** afin de ne pas suivre les changements de certains fichiers.

Il est déjà préconfiguré donc vous n'aurez normalement pas à le modifier non plus.

- Le dossier **test**

Ce dossier contient tous les tests pour notre application Flutter. Par défaut il n'y a qu'un seul fichier qui permet de tester le compteur de l'application par défaut.

- Le dossier **lib**

Ce dossier contient tout le code que nous écrirons pour nos applications Flutter.

Par défaut, il ne contient qu'un seul fichier qui correspond au compteur d'exemple.

- Les dossiers **ios** et **android**

Ces dossiers contiennent tout ce dont nous avons besoin pour générer des applications iOS et android. Nous y reviendrons plus tard dans la formation lorsque nous créerons nos premières versions de production.

- Le dossier **.idea**

Il contient également des fichiers de configuration pour votre éditeur de code. Vous n'aurez pas non plus à les modifier.

?? Lancement de l'application sur Android Studio

Sur Android Studio, lancez simplement l'application en cliquant sur Run :

En mode run, vous pouvez relancer l'application en tapant r dans le terminal ou maj + r pour recharger l'état.

Vous pouvez aussi choisir le mode debug en cliquant sur l'icône à droite de la flèche verte.

Dans ce mode, le hot reload se fait à chaque sauvegarde par défaut (en faisant ctrl + s).

Dans les deux cas l'application est en mode développement, tous les outils de débogage sont activés et disponibles mais l'application est plus lente.

Avec l'application de départ vous aurez sur l'émulateur l'application lancée

:

Vous remarquerez que si vous faites un reload (avec `r` ou avec une sauvegarde suivant le mode), l'état de l'application est maintenu (et donc si vous avez incrémenté le compteur, il conservera sa valeur).