

FORMATION FLUTTER N°1

FLUTTER RÉVOLUTION

Créez de Magnifiques Applications
pour iOS & Android avec Flutter



1. Bienvenue

1.1 Mon histoire

1.2 Mes débuts avec Ionic

1.3 Alors pourquoi Flutter ?

1.4 Pourquoi ce cours ?

2. Qu'est-ce que Flutter ?

2.1 Sa rapidité de développement

2.2 Son interface utilisateur très flexible

2.3 Des performances natives

2.4 Qui utilise Flutter ?

3. Qu'est-ce que Dart ?

3.1 Le Dart et les autres langages

3.2 Le SDK Dart

3.3 Quelques exemples du Dart

4. Installer Flutter sur macOS

4.1 Installer le SDK Flutter

4.2 Configurer le SDK Flutter

4.2.1 Méthode 1: zsh (recommandée)

4.2.2 Méthode 2: bash (utilisée dans la vidéo)

4.3 Configuration de Xcode

4.4 Configurer Android Studio

5. Installer Flutter sur Windows

5.1 Installer le SDK Flutter sur Windows

5.2 Configurer le SDK Flutter sur Windows

5.3 Configurer Android Studio

6. Configurer Visual Studio Code

6.1 Installer et configurer Visual Studio Code

6.2 Créer une application en partant de zéro

1. Bienvenue

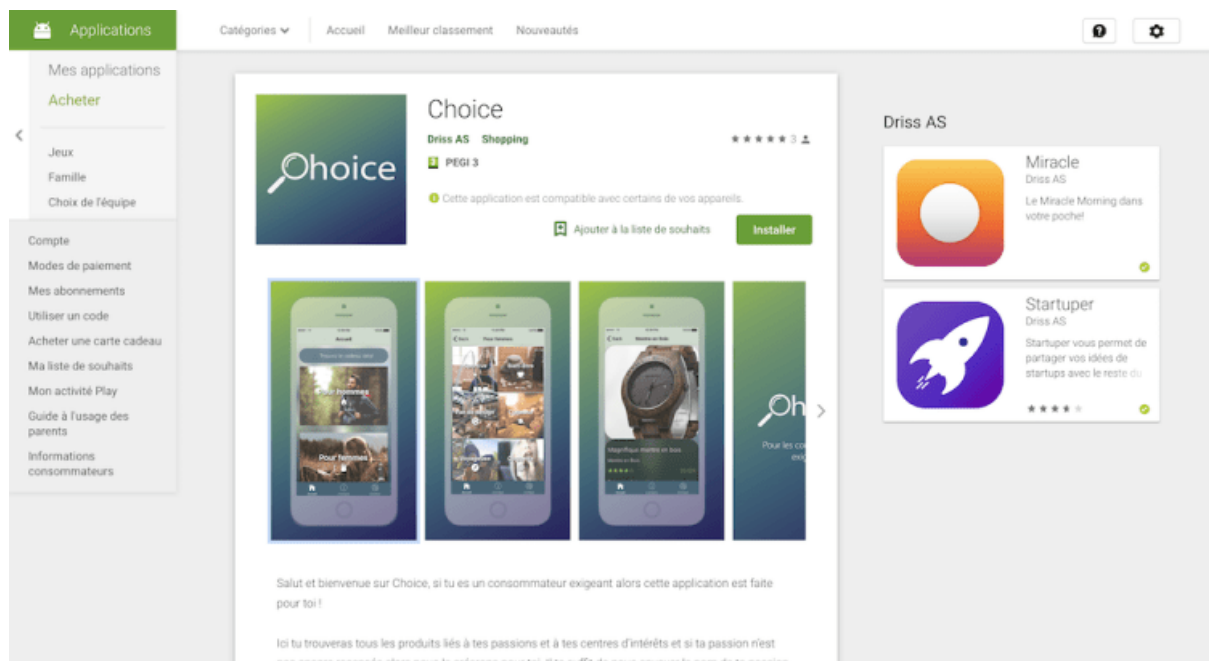
Bonjour à tous et bienvenue dans **FLUTTER RÉVOLUTION!**

Je vous remercie tout d'abord de m'avoir fait confiance et de me rejoindre dans ce cours qui j'en suis sûr **changera votre vision** du développement mobile.

1.1 Mon histoire

Si vous connaissez un peu mon parcours, vous savez que je suis **développeur mobile** depuis **2017** après avoir commencé par le développement web.

J'ai commencé à créer mes premières applications avec le **framework Ionic** qui à l'époque m'a permis de démarrer assez rapidement.



Choice, ma toute première application mobile

J'ai par la suite étendu mes connaissances et proposé **mes services** à d'autres clients, puis finalement j'ai **créé des cours**.

À travers ma chaîne **YouTube** et mon **blog**, vous êtes maintenant des milliers à suivre mes vidéos et mes **tutoriels gratuits** sur le développement mobile.

Mon activité a pu ainsi prospérer grâce à mes **cours en ligne** payants et à mes prestations de cours particuliers.

1.2 Mes débuts avec Ionic

Ionic utilise le framework **Angular de Google** qui est basé sur les technologies **Web** et permet de coder des applications avec les langages **HTML, CSS et TypeScript**.



Le framework Ionic permet lui aussi de développer pour iOS et Android

Le framework le plus connu cependant à l'époque où j'ai commencé à me former était **React JS** de Facebook.

Il avait l'avantage d'être utilisé par de grandes startups dont **Facebook** et **Instagram** par exemple ce qui lui confie une certaine notoriété.

J'ai par la suite longtemps justifié mon choix du framework Ionic en invoquant sa **simplicité**.

En fait Ionic n'a jamais été aussi sexy et **populaire** que React... Mais je suis toujours parti du principe que si j'avais pu **démarrer de zéro** avec Ionic, les autres le pouvait aussi.

1.3 Alors pourquoi Flutter ?

J'avais déjà annoncé à plusieurs reprises que je passerai un jour ou l'autre à **React JS** pour m'adapter à la forte demande du **marché**.

Pourtant, vous êtes en train de lire un **cours sur Flutter**... Alors pourquoi ?

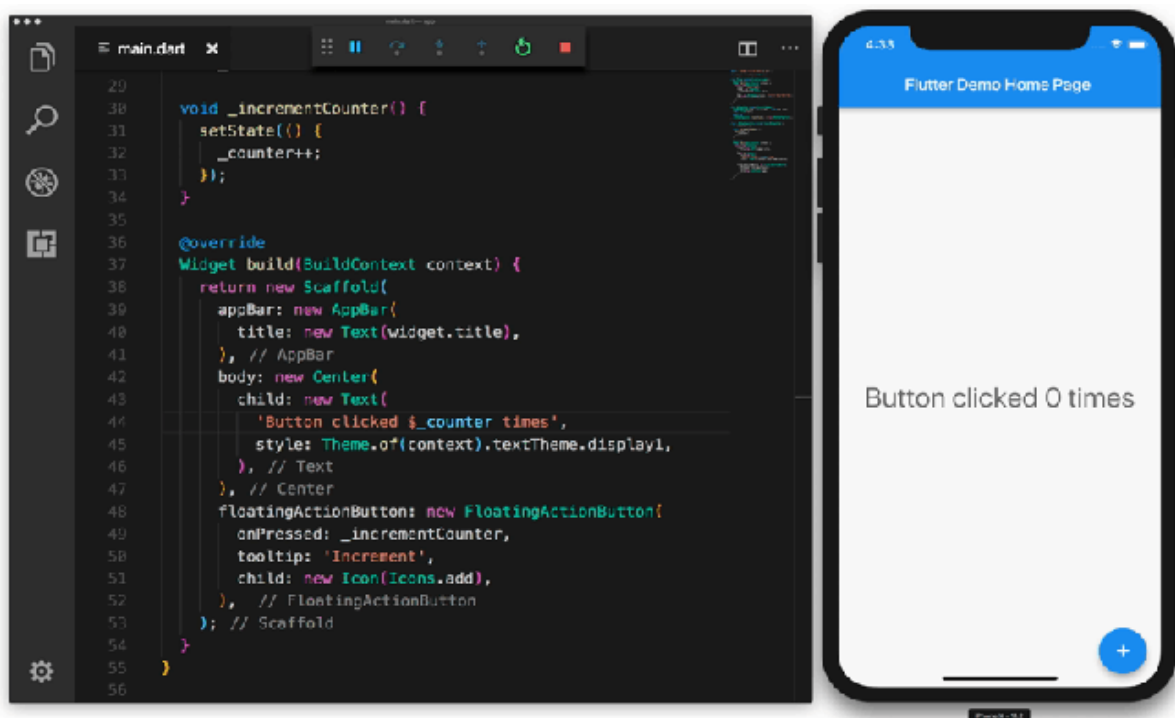
Je pourrais passer très longtemps à répondre à cette question, le **développement mobile est mon quotidien** et je baigne en permanence dans toute l'actualité des startups.

Mais pour faire simple, c'est parce que **Flutter** représente selon moi une véritable **révolution**...

Ce n'est pas qu'un simple **jeu de mot** pour nommer cette formation, c'est véritablement ce que j'ai **ressenti en testant pour la première fois Flutter**.

Flutter permet de développer **très rapidement** par rapport à Ionic ou React, il est directement orienté vers le développement **natif** sur émulateur **iOS et Android**.

En fait Flutter est un **SDK** et non un framework comme React, ce qui veut dire qu'il s'adapte en fait aux environnements de développement d'Android et iOS via leurs logiciels respectifs.



Flutter avec un émulateur iOS, s'actualise instantanément

Concrètement avec **un seul code** (le **Dart**) et simplement en enregistrant son fichier on actualise son application sur iOS et Android.

Cette simple différence par rapport aux frameworks JS est en fait monstrueuse, pas pour nos applications directement, mais pour notre **rapidité de développement**.

On développe tout simplement **plus vite** sur Flutter que sur n'importe quel **autre framework** multiplateforme.

1.4 Pourquoi ce cours ?

J'ai donc décidé en **juillet 2020** de démarrer un nouveau et unique cours sur Flutter.

Pour aider la **communauté francophone** du monde entier à exploiter au maximum le superbe outil que représente Flutter.

Dans ce cours, nous allons reprendre toutes les **bases de Flutter** et du **Dart** en repartant vraiment **de zéro**.

Le but pour moi est de vous permettre d'aller **le plus loin possible** dans le développement de vos applications mobiles avec Flutter.

On commencera avec la prise en main des **bases** de Flutter et de la maîtrise de son **langage Dart**.

Je vous proposerai aussi plusieurs chapitres consacrés aux **interfaces utilisateurs** que propose Flutter qui sont à la pointe des **animations** sur smartphone.

Enfin on terminera par un ensemble de **tutoriels ciblés** sur les **fonctionnalités** les plus utiles des applications modernes.

2. Qu'est-ce que Flutter ?

Flutter est en fait un **SDK** et non un framework de développement mobile. Qu'est-ce qu'un SDK par rapport à un framework et qu'est-ce que cela représente-t-il pour nos applications ?

Selon **Wikipédia**, la définition officielle d'un Framework est la suivante:

En programmation informatique, un **framework** désigne un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel.

<https://fr.wikipedia.org/wiki/Framework>

Concrètement un **framework** comme Angular ou React permet de **créer de A à Z** une application mobile par exemple.

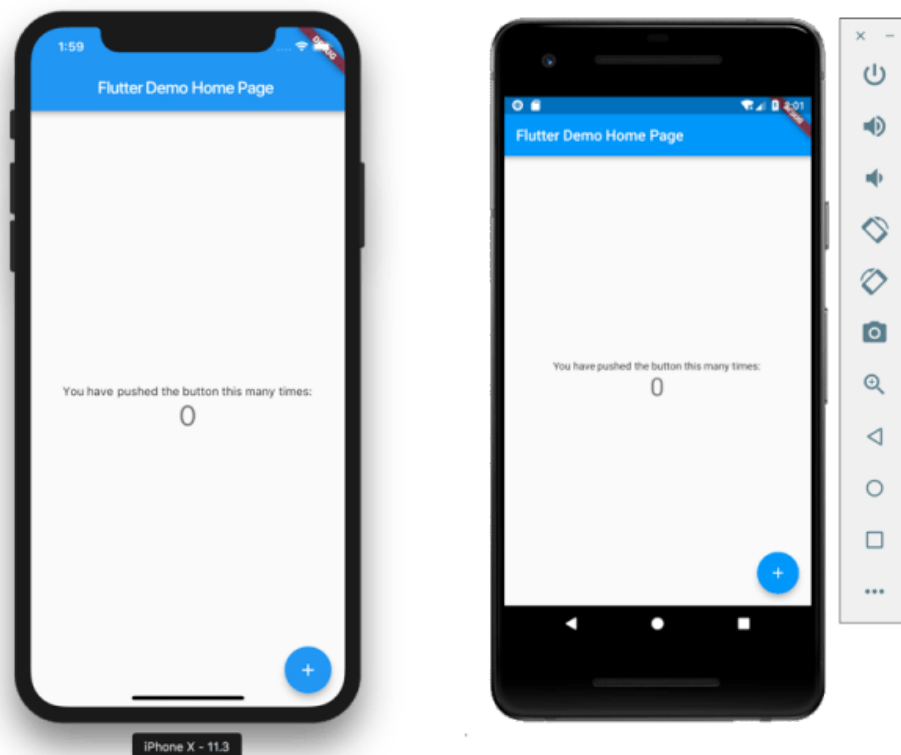
Il nous faut cependant passer par une **phase d'adaptation** aux appareils mobile lors de la création de chaque version iOS et Android.

Flutter en revanche est donc un **SDK**, qui veut dire littéralement **Kit de Développement** (*Software Development Kit*).

Google a conçu le **SDK Flutter** pour s'adapter aux deux environnements de développements iOS et Android: **Xcode et Android Studio**.

Le SDK Flutter vient donc s'installer sur votre machine en **complément** de ces deux logiciels (avec Android qui comprend aussi son propre SDK).

Donc Flutter nous permet de travailler depuis un **éditeur de code** et de compiler pour les deux plateformes en même temps.



Flutter permet de lancer deux émulateurs iOS et Android en même temps

Voyons donc maintenant quels sont les **points essentiels** à connaître sur l'utilisation de Flutter.

2.1 Sa rapidité de développement

Le premier **point essentiel** encore une fois de Flutter est la rapidité de développement qu'il nous offre.

En fait Flutter nous offre une rapidité de développement que l'on retrouve avec le **développement natif** d'application.

Sur **Xcode** lorsqu'on code avec le **Swift** et qu'on teste son application en quelques secondes avec un simulateur **d'iPhone**.

Mais aussi sur **Android Studio** avec **Java ou Kotlin** lorsqu'on teste son application sur un émulateur **Android**.

En fait avec Flutter on utilise un **éditeur de code** (comme Visual Studio Code dont nous détaillerons la configuration) pour travailler sur nos fichiers **Dart**.

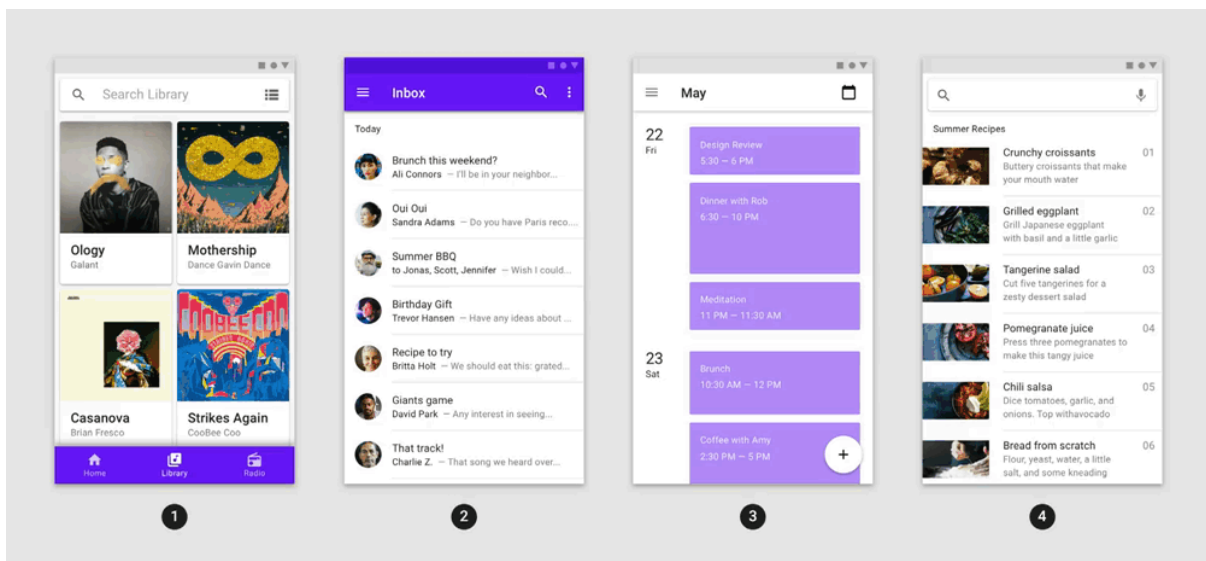
À chaque **enregistrement** de nos fichiers, grâce à l'extension Flutter de Visual Studio notre **application s'actualise** sur notre émulateur.

2.2 Son interface utilisateur très flexible

Le deuxième point essentiel de Flutter est son **interface utilisateur** ou UI Design vraiment **révolutionnaire**.

En fait la plupart des autres frameworks ou outils de développement mobile permettent de recréer des **animations modernes**.

Mais Flutter a vraiment mis une **priorité** sur ces animations pour les rendre très **accessibles**.



Nous verrons par exemple comment créer avec les **Widgets Flutter** toute sorte d'affichage pour nos applications.

Mais nous utiliserons aussi l'immense **bibliothèque d'animations** et d'interactions que Flutter propose pour **dynamiser** nos applications.

À la différence d'autres frameworks de développement mobile, ses interactions sont **facilement accessibles**.

C'est la raison pour laquelle la grande majorité des **applications** développées avec Flutter sont très **intuitives** et dynamiques.

Vous pouvez en retrouver quelques **exemples** sur le site de Flutter:
<https://flutter.dev/showcase>

2.3 Des performances natives

Je vous ai parlé à plusieurs reprises de la rapidité de **développement** que propose Flutter, mais parlons maintenant de la **rapidité de nos applications** elle-même.

Il est évident que **Google a conçu Flutter** pour proposer la meilleure rapidité d'exécution possible pour ses applications, mais à quel point une application peut-elle être **plus rapide** ?

En fait quand on parle de **performance native** on parle notamment de l'accès aux fonctionnalités natives d'un smartphone.

Avec Ionic, nous utilisons **Cordova JS** pour accéder aux fonctionnalités natives d'un appareil mobile, comme **l'appareil photo** ou la **localisation**.

Avec Flutter, qui je vous le rappelle est un **SDK**, on accède beaucoup plus rapidement aux **fonctionnalités natives**.

Google prétend même que les performances sont **équivalentes au développement natif** d'application avec Swift et Java notamment.

À ce stade, c'est assez **difficile à vérifier**, car les derniers modèles d'iPhone sont déjà extrêmement performants et les différences de performance se calculent en millisecondes.

Même les applications de Facebook comme **Instagram** développées avec React proposent des performances excellentes.

Nous pouvons donc **faire confiance à Google** en ce qui concerne les performances et le maintien de celle-ci dans la durée.

2.4 Qui utilise Flutter ?

Le SDK Flutter est **le plus récent** de tous les outils logiciels du monde du développement mobile.

Comme je vous l'ai dit, je me suis spécialisé pendant des années sur le framework Ionic mais qui était lui-même basé sur **Angular**, un autre framework de **Google**.

L'avantage d'utiliser un **framework mature** est que l'on retrouve beaucoup de **ressources sur internet** pour nous faciliter l'apprentissage.

Sur **Flutter**, les cours en ligne et tutoriels en français restent **plutôt rares** pour le moment d'où cette formation.

En ce qui concerne les **entreprises** qui utilisent Flutter, les plus connus se comptent pour le moment sur les doigts d'une main.



Vous retrouverez toutes les **startups** qui l'utilisent sur le site de Flutter:
<https://flutter.dev/showcase>

Mais pas d'inquiétude, à la **fin de ce cours** vous ferez peut-être partie des futurs startups ou développeurs **sélectionnés par Google** pour mettre en avant Flutter 😊

3. Qu'est-ce que Dart ?

Contrairement à ce qu'on peut penser, le projet du langage **Dart** chez Google est **plus ancien** que le projet **Flutter**.

Il a été développé chez Google au départ pour combler les défauts du **JavaScript** et avait pour objectif de le remplacer en tant que langage phare du web.



Flutter n'est venu dans un deuxième temps pour trouver une **nouvelle utilité** au langage Dart resté inconnu du grand public.

Aujourd'hui avec le langage Dart et Flutter, nous pouvons créer des **applications mobiles**, des **sites web** et des **logiciels de bureau**.

Dart est donc un langage très **puissant** et **polyvalent**, car il permet de gérer le côté **design** avec Flutter mais également la **connexion** avec **Firebase**.

Bref le langage Dart vous offrira de nombreux **choix** en termes de développement et vous ouvrira aussi des portes pour votre **carrière**.

3.1 Le Dart et les autres langages

Le Dart on va le voir n'est pas si nouveau que ça, mais vient parfaitement **s'adapter aux besoins** des développeurs **d'applications**.

Le Dart est un **langage orienté objet** comme le sont la plupart des langages de développement logiciel (C#, Swift, ou Objective C).

On sera amené par exemple à le **comparer** au **JavaScript** qui fait tourner la majorité des autres frameworks multiplateformes.

Contrairement à Angular par exemple, on développe **quasiment 100%** de notre **code** avec le **Dart**.

Sur les frameworks JavaScript comme **Angular et React**, on développe avec les **langages web** classiques:

- **HTML**: Pour le contenu de notre application (texte, image, vidéo, boutons)
- **CSS**: Pour styliser nos pages et ajouter des animations

- **JavaScript:** Pour dynamiser nos applications et les connecter aux bases de données



Avec Ionic,

on développe en HTML, CSS et JS

Avec Flutter, **98%** de votre application sera **codée en Dart**, qu'il s'agisse de la partie **design** ou de la connexion à vos bases de données Firebase.

C'est à la fois un **avantage et un inconvénient**, car dans le cas des langages web cela facilite l'apprentissage au début.

Le langage **HTML** par exemple est **très facile** à prendre en main, et on peut progressivement styliser nos pages avec le CSS puis les dynamiser avec le JS.

Avec Flutter et **Dart** c'est différent, on doit obligatoirement **comprendre et maîtriser** le Dart pour afficher du contenu.

L'avantage est qu'une fois le Dart maîtrisé, on peut **tout coder** avec ce même langage jusqu'à la fin du développement de notre application.

Nous allons maintenant parler plus en détail du **SDK du Dart** dont je vous laisse le site web: <https://dart.dev/>

3.2 Le SDK Dart

Tout comme Flutter, Dart est proposé par Google **sous forme de SDK** à télécharger et installer sur votre machine.

Il se télécharge **automatiquement** avec l'installation de **Flutter** sur votre ordinateur.

Nous détaillerons les **étapes d'installation** sur Mac et Windows dans les **prochains chapitres**.

Celui-ci contient toutes les **bibliothèques Dart** que nous utiliserons pour développer nos applications.

3.3 Quelques exemples du Dart

Reprenons **quelques exemples** de code Dart pour comprendre les spécificités de ce langage.

Ces exemples sont tirés de la **documentation de Google** sur le site <https://dart.dev/#try-dart>.

On commence avec la traditionnelle fonction **Hello Word** qui permet d'afficher ce texte dans la console:

```
void main() {  
  print('Hello, World!');  
}
```

Le langage Dart a toujours besoin d'une **fonction principale *main()*** pour exécuter le reste de son code.

On retrouvera ainsi pour chacun de nos codes Dart **une unique fonction** `main()` qui appelle d'autres fonctions à son tour.

Ici on utilise la fonction d'affichage ***print()*** pour afficher dans la **console** le message: *"Hello, World!"*.

On peut également afficher d'autres **chaînes de caractères** avec une simple fonction `print()`:

```
void main() {  
  print('a single quoted string');  
  print("a double quoted string");  
}
```

```
// Strings can be combined with the + operator.
print('cat ' + 'dog');
// Triple quotes define a multi-line string.
print('''triple quoted strings
are for multiple lines''');
}
```

On utilisera généralement cette fonction **print()** pour **afficher** ou récupérer des **informations** à certains endroits de notre code pour vérifier son bon fonctionnement.

On peut également déclarer des **constantes** et les intégrer à notre fonction **print()** avec la syntaxe **\$constante**:

```
void main() {
  // Dart supports string interpolation.
  const pi = 3.14;
  print('pi is $pi');
  print('tau is ${2 * pi}');
}
```

On finit par voir un exemple de déclaration de fonction en Dart pour **afficher** un **nombre** dans la **console**:

```
// Define a function.
void printInteger(int aNumber) {
  print('The number is $aNumber.');// Print to console.
}

// This is where the app starts executing.
void main() {
  var number = 42; // Declare and initialize a variable.
  printInteger(number); // Call a function.
}
```

On déclare la fonction **printInteger()** en lui indiquant le type de paramètre à prendre (ici **int** représente les variables de types nombres).

Ensuite dans notre fonction **main()**, on peut faire appel à cette fonction **printInteger()** en lui indiquant un paramètre de type nombre.

4. Installer Flutter sur macOS

Dans ce chapitre, nous allons voir comment installer le **SDK Flutter** sur votre **Mac** de A à Z.

Une distinction est faite entre les **systèmes d'exploitation**, car les méthodes de téléchargement sont différentes entre Mac et PC.

Sur **MacOS**, vous avez la possibilité d'Installer **Xcode** et **Android Studio** et vous serez donc capable de développer pour **iOS** et **Android**.

Sur **Windows** le logiciel **Xcode** d'Apple n'est **pas disponible** et vous ne pourrez pas développer ou publier sur iOS.

Vous pouvez néanmoins faire par exemple tourner une **machine virtuelle** sur votre PC avec MacOS.

Passons maintenant aux **étapes d'installation** de Flutter sur MacOS dont je vous laisse la documentation: <https://flutter.dev/docs/get-started/install/macos>

4.1 Installer le SDK Flutter

Commencez par **télécharger le fichier ZIP** contenant tout le SDK Flutter à installer sur votre ordinateur:

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

Get the Flutter SDK

1. Download the following installation bundle to get the latest stable release of the Flutter SDK:

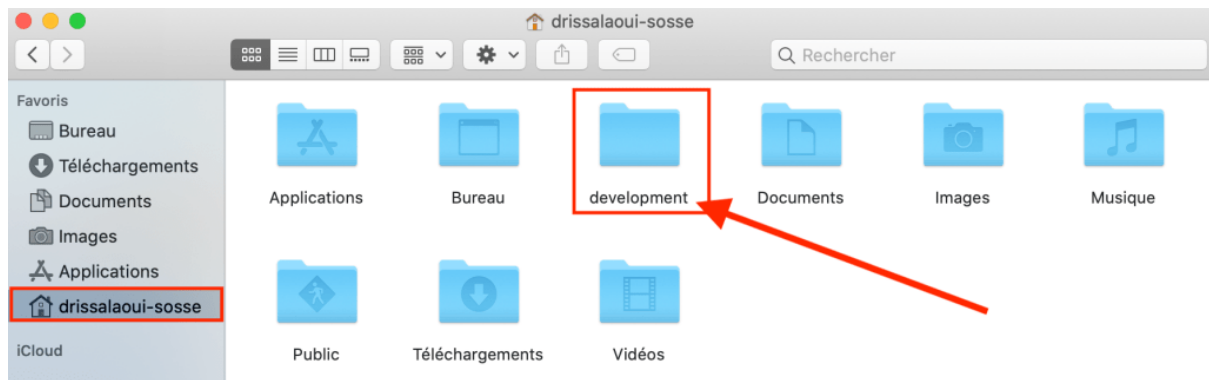
[flutter_macos_1.17.5-stable.zip](#)



For other release channels, and older builds, see the [SDK archive](#) page.

Nous allons reprendre toutes les étapes de l'installation et suivre au maximum les **conseils** que propose la **documentation de Flutter**.

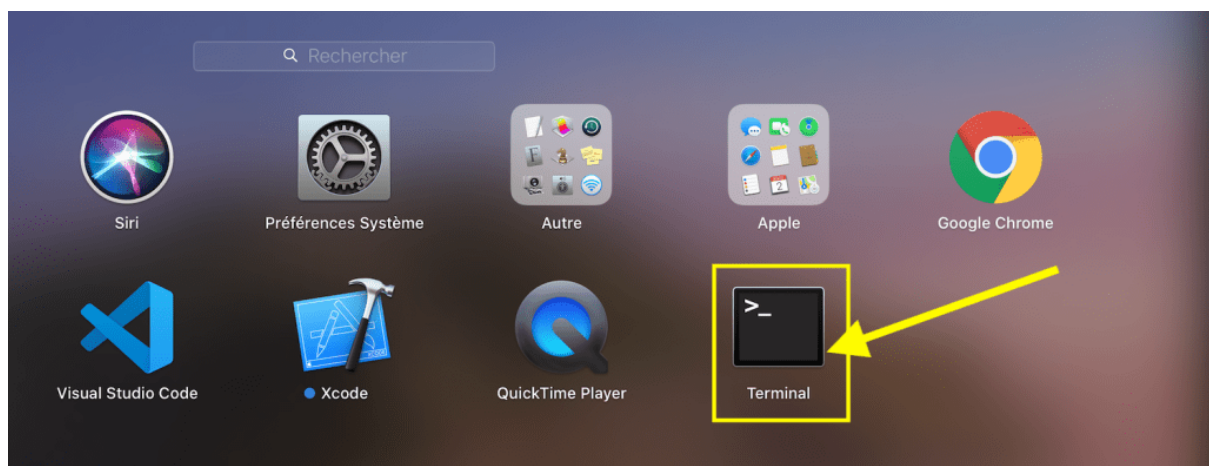
Ainsi, je vous invite à créer comme il le recommande un **dossier** "**development**" à la racine de votre **session utilisateur**:



Par exemple dans le dossier de ma **session** “drissalaoui-sosse”, je crée le dossier **development**.

Ce nouveau dossier development contiendra le **SDK de Flutter** ainsi que toutes vos **futures applications** Flutter.

Pour **installer** définitivement le **SDK Flutter** sur votre Mac nous allons utiliser le **Terminal**, cherchez donc le logiciel sur votre Mac et ouvrez-le:



Une fois ouvert votre terminal devrait s’ouvrir au **dossier racine** de votre **session utilisateur**:



Vous pouvez ensuite vous **rendre** dans le dossier **development** que vous devriez avoir créé précédemment:

```
cd development
```

Une fois situé dans le dossier **development** dans votre terminal, nous allons pouvoir **dézipper** le fichier télécharger sur le site de Flutter.

Vous pouvez le **dézipper** manuellement ou utiliser la **commande** suivante pour dézipper le fichier **flutter_macos_2.2.3-stable.zip** (*attention le nom du fichier change avec les versions*):

```
unzip flutter_macos_2.2.3-stable.zip
```

Prenez soin auparavant de situer votre **fichier zip** dans le **dossier development**.

Voilà pour l'**installation** du SDK Flutter, nous allons maintenant faire en sorte de le **configurer** pour un fonctionnement optimal.

4.2 Configurer le SDK Flutter

Nous venons donc **d'installer le SDK Flutter** sur notre Mac, nous avons ainsi accès aux **commandes Flutter** pour créer de nouvelles applications par exemple.

Néanmoins, nous avons besoin de **configurer l'adresse de notre SDK** pour qu'il soit accessible partout sur notre Mac.

Le **SDK** se trouve précisément dans le **dossier *bin*** de notre **dossier flutter**, nous allons donc stocker dans notre Mac **l'adresse exacte** de ce dossier.

On utilise la **commande *pwd*** pour **trouver** l'adresse complète du dossier ***flutter/bin***.

Vous pouvez ainsi utiliser les **commandes** ci-dessous pour déclarer et **afficher** cette **nouvelle adresse**:

```
export PATH="$PATH:`pwd`/development/flutter/bin"
echo $PATH
```

Copiez et mettez l'adresse affichée par le terminal **de côté** pour pouvoir la **réutiliser** plus tard.

Je vais vous proposer **deux méthodes** pour déclarer votre **variable globale** d'accès au SDK Flutter.

4.2.1 Méthode 1: zsh (recommandée)

Je commence par vous proposer la **méthode** la **plus récente**, qui utilise **l'interpréteur de commande zsh** pour Mac.

Dans la **vidéo**, je vous montre l'exemple avec l'interpréteur **bash**, le principe est le même sauf que le zsh est le **successeur** de bash.

Reprenez votre **terminal** et entrez la commande:

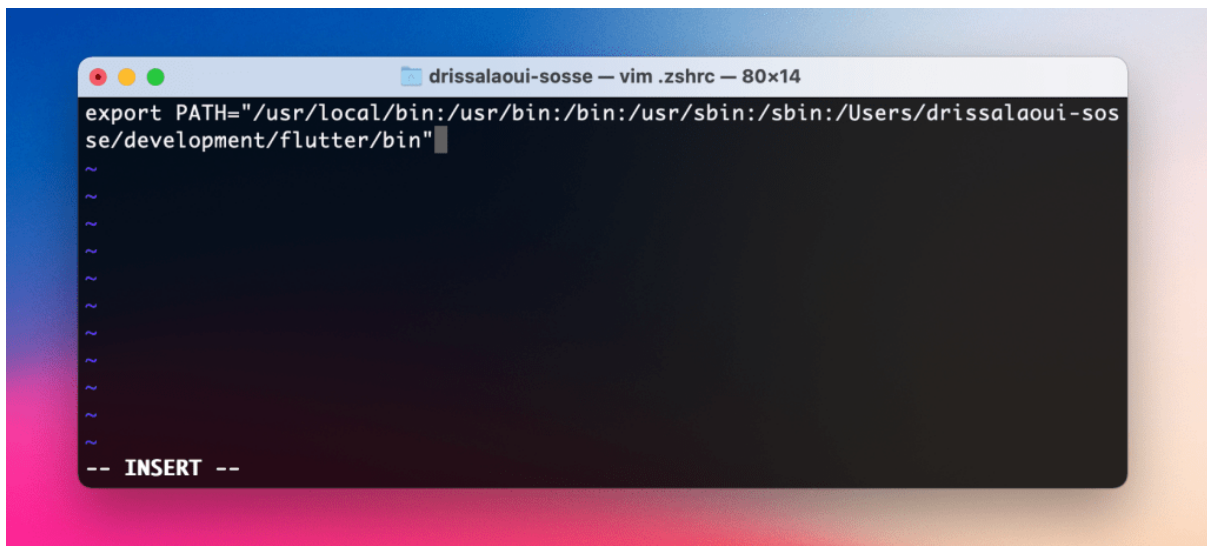
```
vim $HOME/.zshrc
```

Dans le **fichier** qui s'affiche dans votre terminal, vous allez pouvoir déclarer l'adresse du SDK Flutter et pouvoir ainsi y accéder depuis tout votre ordinateur.

Reprenez donc **l'adresse** de votre dossier ***flutter/bin*** et **déclarez la variable** suivante:

```
export PATH="L_ADRESSE_DU_DOSSIER_FLUTTER_BIN"
```

Voilà par **exemple** ce que donne le rendu de mon **fichier zshrc**:



```
drissalaoui-sosse — vim .zshrc — 80x14
export PATH="/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Users/drissalaoui-sosse/development/flutter/bin"
~
~
~
~
~
~
~
~
-- INSERT --
```

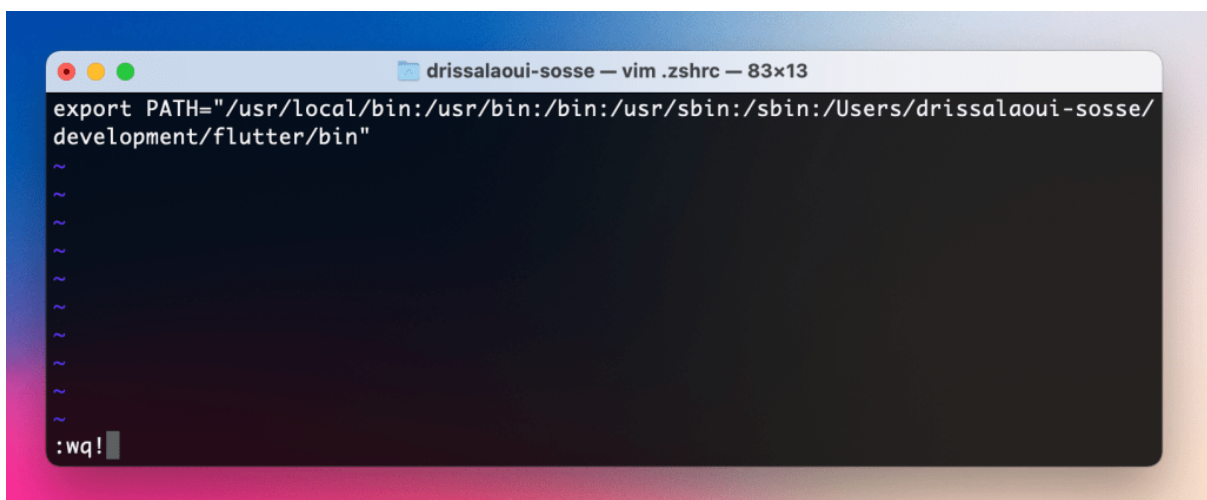
Pour **quitter** l'édition du fichier, entrez:

```
Crtl + C
```

Puis tapez dans le terminal les caractères suivants pour quitter et **enregistrer votre fichier**:

```
:wq!
```

Cette commande (**write and quite**) vous permet de bien enregistrer votre fichier, devrait être visible en bas de votre terminal:



```
drissalaoui-sosse — vim .zshrc — 83x13
export PATH="/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Users/drissalaoui-sosse/development/flutter/bin"
~
~
~
~
~
~
~
~
:wq!
```

Une fois revenu à votre terminal habituel, n'oubliez pas de **REDÉMARRER** votre logiciel avant de tester que la **méthode zsh** a bien **fonctionné**.

Pour tester que votre **variable PATH** a bien été mise à jour, vous pouvez entrer la commande suivante pour **l'afficher**:

```
echo $PATH
```

Vous pouvez aussi **tester** par exemple la commande ***flutter doctor*** pour vérifier si la commande flutter est accessible (ne faites pas encore attention au résultat de la commande):

```
flutter doctor
```

Je vous montre maintenant la **deuxième méthode**, que j'utilise dans la vidéo pour ceux que ça intéresse.

4.2.2 Méthode 2: bash (utilisée dans la vidéo)

Si vous le souhaitez, vous pouvez aussi conserver la **méthode bash**, qui est similaire, mais **moins recommandé** pour Mac à ce jour.

Nous allons utiliser le fichier **bash_profile** pour stocker cette adresse de manière définitive.

Entrez les **deux commandes** suivantes pour **créer** puis **ouvrir** le fichier **bash_profile** (vous devez vous trouver dans votre dossier racine utilisateur):

```
touch .bash_profile  
open .bash_profile
```

Un **document texte** vierge devrait s'ouvrir, vous pouvez alors lui **entrer** la ligne de code suivante:

```
export PATH=L_ADRESSE_DU_DOSSIER_FLUTTER_BIN
```

Par **exemple** voilà ce que j'entre dans **mon fichier** bash_profile:

```
export  
PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Users/dris  
salaoui-sosse/development/flutter/bin
```

Enregistrez ensuite votre fichier texte et **quittez l'éditeur** TextEdit, entrez ensuite la **commande source** suivante pour appliquer les modifications.

```
source .bash_profile
```

Vous pouvez ensuite **tester** par exemple la commande ***flutter doctor*** pour tester si la commande flutter est accessible (ne faites pas encore attention au résultat de la commande):

flutter doctor

On passe maintenant à la **configuration de Xcode** et du lancement de notre émulateur d'iPhone.

4.3 Configuration de Xcode

Nous allons maintenant **installer** le logiciel de développement d'Apple **Xcode** pour accéder aux **émulateurs d'iPhone**.

Vous pouvez **télécharger Xcode** gratuitement en vous rendant sur le **Mac App Store**: <https://apps.apple.com/fr/app/xcode/id497799835?mt=12>



Nouveautés

Xcode 11.6 supports developing apps for iOS 13.6, iPadOS 13.6, tvOS 13.4, watchOS 6.2, and macOS Catalina 10.15.6

[suite](#)

[Historique](#)

Il y a 2 sem
Version 11.6

Aperçu



Xcode est un logiciel assez **volumineux** qui demande chaque année une **mise à jour**.

Votre version de **MacOS** doit aussi **être à jour** pour permettre à Xcode de vous faire utiliser les dernières **fonctionnalités d'iOS**.

Si ce n'est pas déjà fait, mettez donc à **jour** votre version de **MacOS** et installer **Xcode**.

Une fois le logiciel installé sur votre Mac, entrez les **commandes** pour **configurer Xcode** sur votre terminal:

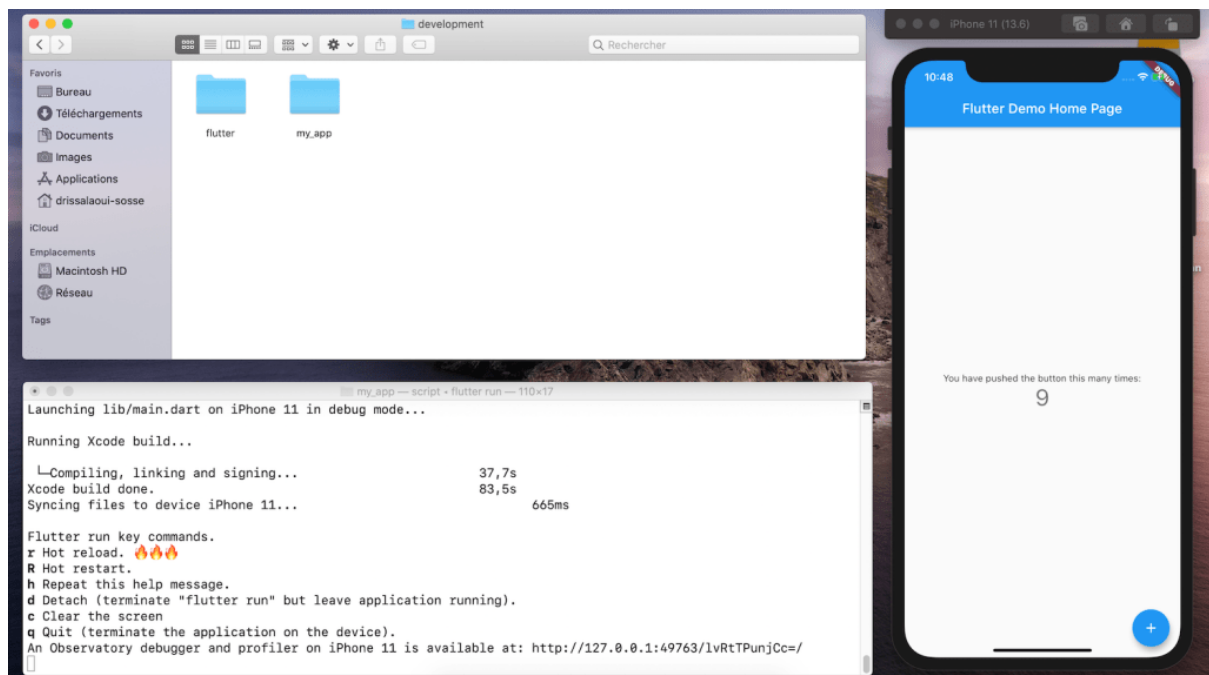
```
sudo xcode-select --switch
/Applications/Xcode.app/Contents/Developer
sudo xcodebuild -runFirstLaunch
```

Le mot clé **sudo** vous indique que vous devez entrer votre **mot de passe** dans votre terminal pour valider cette action.

Une fois ces deux commandes entrées vous pouvez lancer un premier **simulateur d'iPhone 8** avec la commande **open**:

```
open -a Simulator
```

Vous pouvez **changer d'appareil iOS** en fait un **clic-droit** sur l'icône de votre émulateur dans dock et sélectionner par exemple un **iPhone 11**.



Voilà donc le **résultat** de votre émulateur iOS qui fonctionne et avec lequel nous allons pouvoir tester nos **applications Flutter**.

Si vous **créez** une **première application** rapidement et la **lancer** dans votre émulateur, entrez les **commandes suivantes**:

```
flutter create my_app
```

```
cd my_app  
flutter run
```

Nous détaillerons ces **étapes** dans un **autre chapitre**.

4.4 Configurer Android Studio

La configuration d'**Android Studio** est plus **rapide**, mais **pas moins technique** que pour Xcode.

Je vous invite encore une fois à vous référer en parallèle à ce chapitre à la **documentation de Flutter**:

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

Vous pouvez notamment commencer par **télécharger Android Studio** sur Mac à cette adresse: <https://developer.android.com/studio>



Android Studio provides the fastest tools for building apps on every type of Android device.

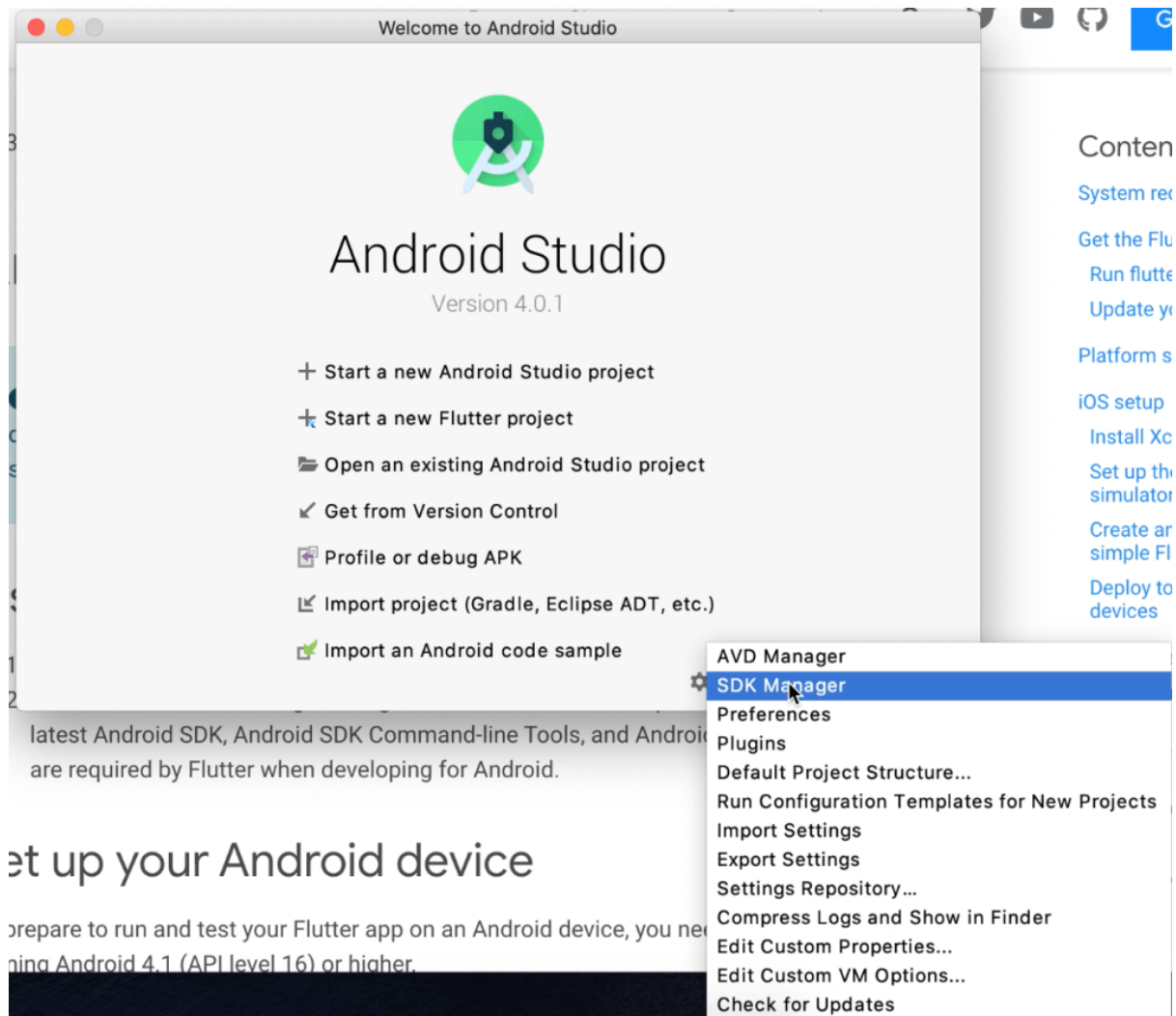


4.0.1 for Mac (856 MB)

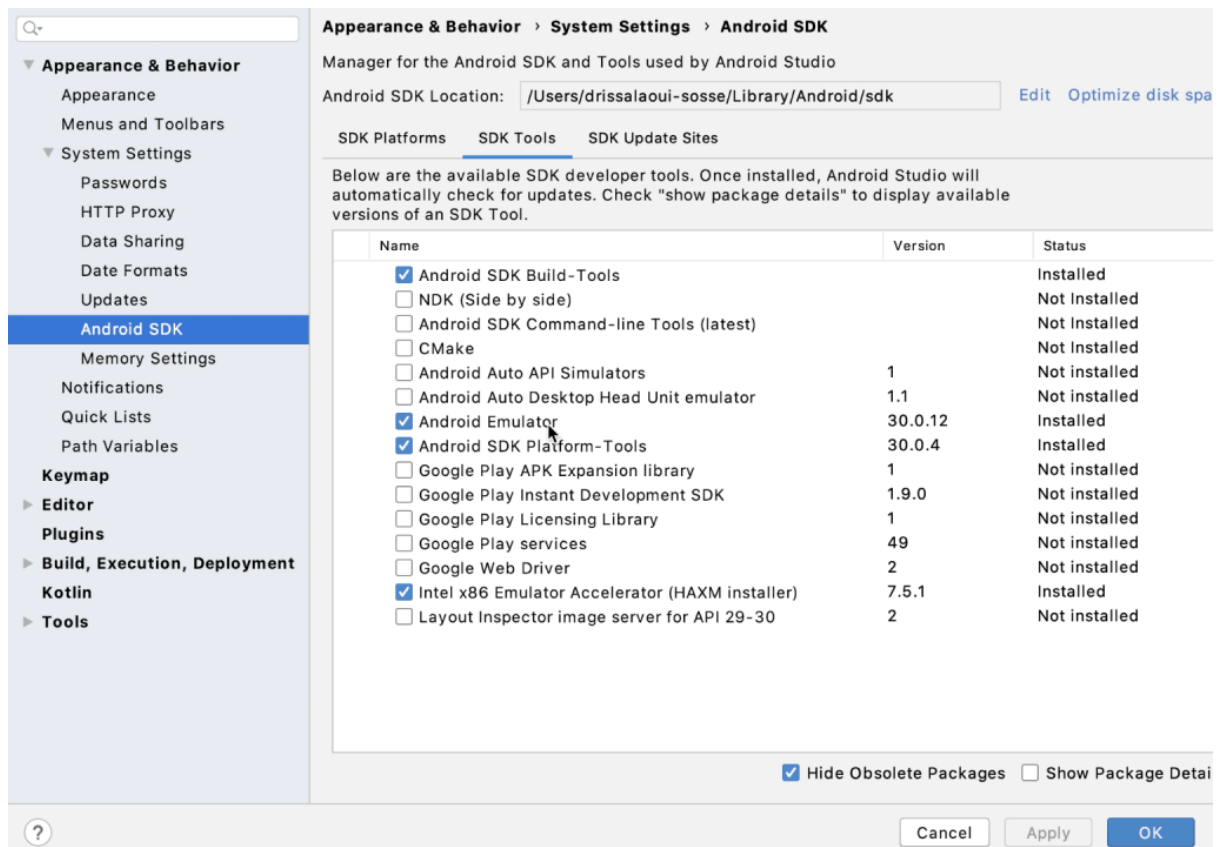
Lancez ensuite le **fichier téléchargé** pour démarrer l'**installation** d'Android Studio.

Vous devriez voir également les différentes **SDK supplémentaires** qui vont être **installés** par la même occasion.

Une fois installé, **lancez Android Studio** une première fois et ouvrez les paramètres **SDK Managers**:

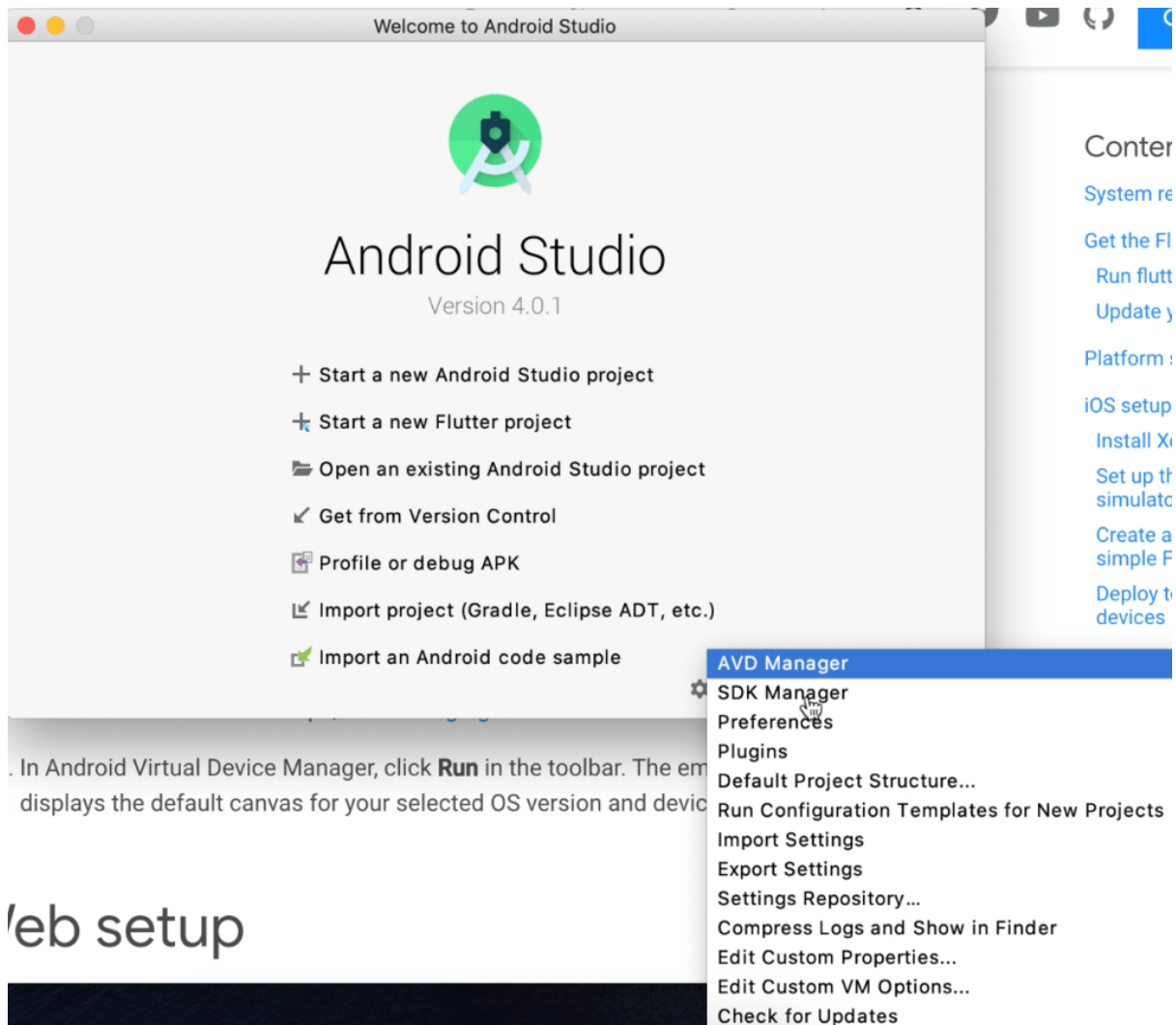


Vous pouvez de cette manière **vérifier** la présence du **SDK Android Build Tool** ainsi que du **Android emulator** que nous allons utiliser.

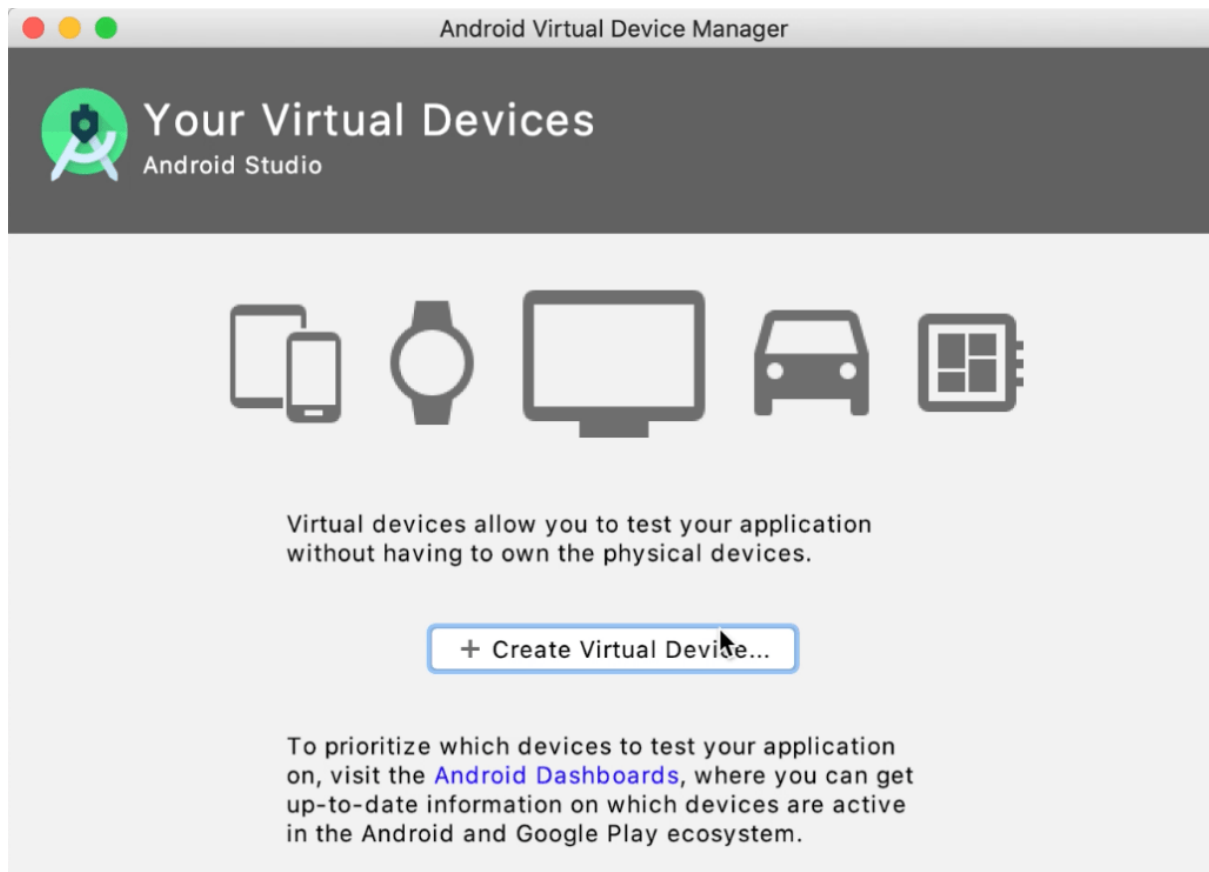


Si jamais il **manque** l'un des outils ou SDK, **cochez-le** pour l'installer sur votre Mac.

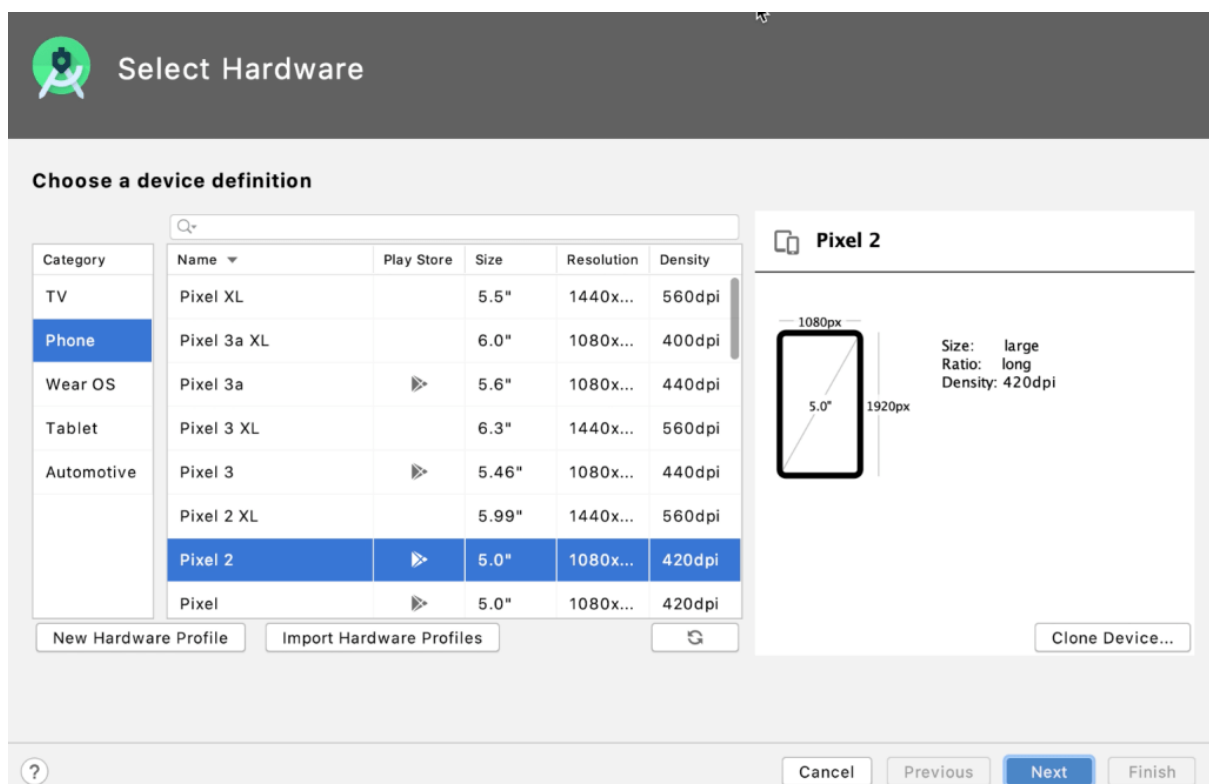
Passons maintenant à la **configuration** d'un **premier émulateur Android**, pour cela ouvrez le menu **AVD Manager**:



L'interface vous proposera lors de **créer un nouvel appareil virtuel**:



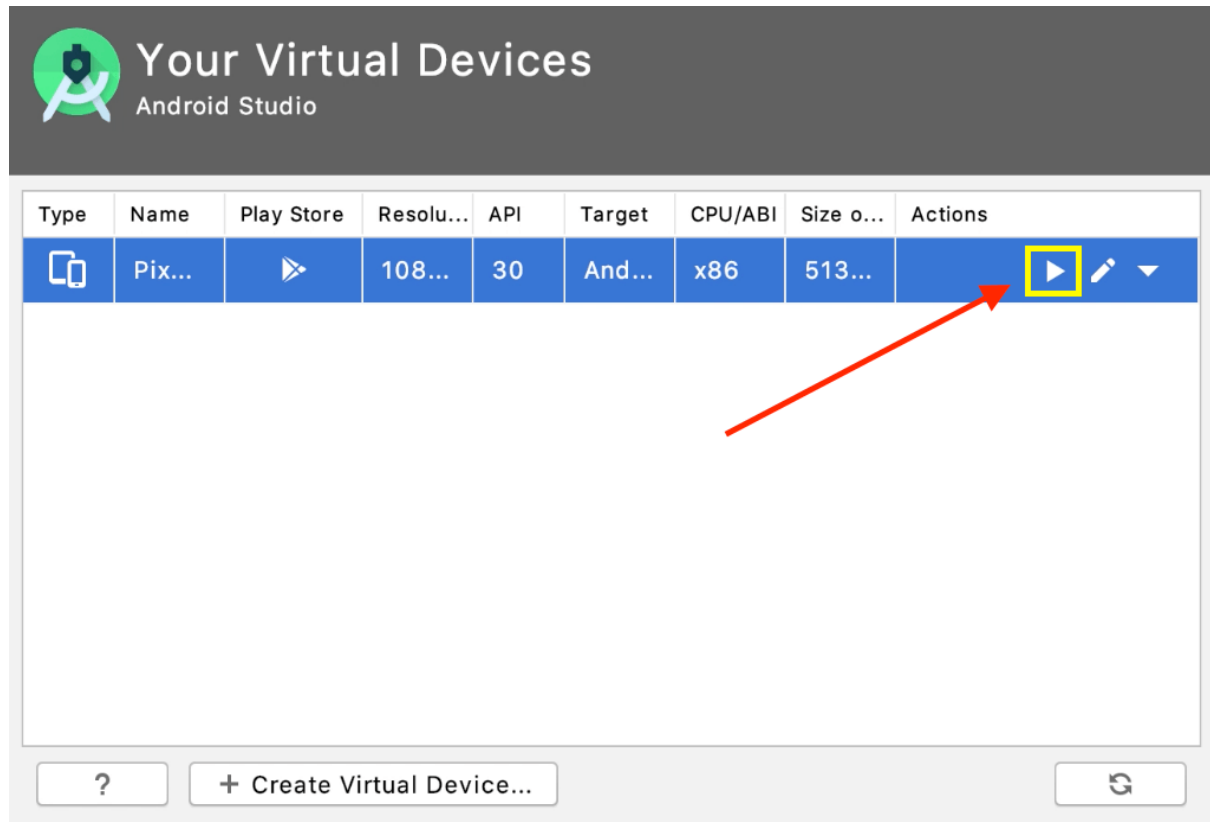
Sélectionnez dans la **catégorie Phone** le modèle de votre choix, par exemple un **Pixel 2** de Google:



Installer et télécharger la **dernière version d'Android OS** sur cet émulateur pour terminer la configuration de cet appareil.

Une fois l'**émulateur** Android **téléchargé** et installé, il devrait apparaître dans le tableau de vos **machines virtuelles**.

Vous pouvez cliquer sur le **bouton Play** pour **lancer** cet émulateur Android.



Un **Pixel 2** devrait donc **apparaître** sur votre Mac, dans lequel nous allons pouvoir **lancer notre première application** Flutter.

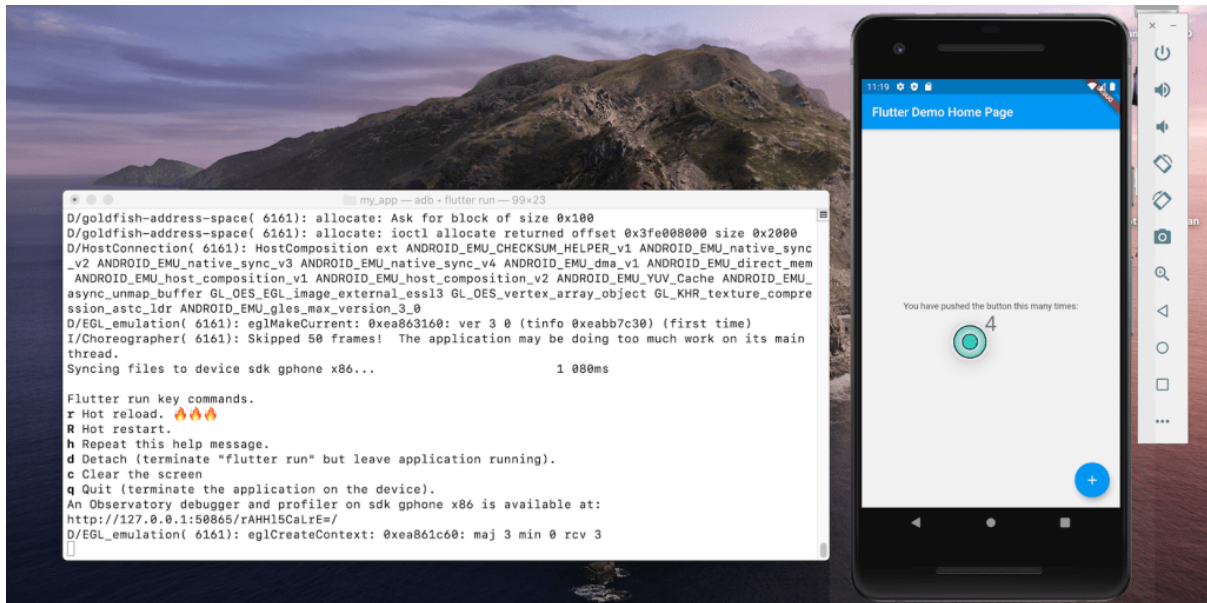
Reprenez votre **terminal** et entrez la commande suivante pour **repérer d'éventuels problèmes** à corriger:

```
flutter doctor
```

Si vous n'avez pas encore créé d'**application Flutter**, vous pouvez en **créer une rapidement** et la lancer dans votre **émulateur Android** avec les commandes suivantes:

```
flutter create my_app
cd my_app
flutter run
```

Votre **application Flutter** devrait ainsi se lancer dans un **Pixel 2** avec l'apparence et le design d'une **application Android** classique.



Voilà pour l'installation de Flutter sur votre Mac.

5. Installer Flutter sur Windows

Dans ce chapitre nous allons voir comment **installer Flutter** sur un PC **Windows**.

La procédure est légèrement **différente** que sur un **Mac**, notamment en ce qui concerne la déclaration des **variables d'environnement**.

En ce qui concerne **Android Studio** le logiciel est **identique** sur Windows et Mac donc pas de différence majeure à ce niveau-là

En revanche sur Windows vous n'avez **pas accès** au **logiciel Xcode** et donc au développement sur **iOS**.

Vous ne pourrez pas tester vos applications sur un émulateur **d'iPhone** ou **d'iPad**.

Pour cela vous devrez soit **installer MacOS** sur votre PC ou le faire tourner sur une **machine virtuelle** avec un logiciel adapté.

5.1 Installer le SDK Flutter sur Windows

Pour commencer nous allons télécharger et installer le **SDK Flutter** sur notre PC **Windows**.

Vous pouvez vous rendre sur le **site de Flutter** pour **télécharger** la dernière version du SDK:

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

🔗 Get the Flutter SDK

1. Download the following installation bundle to get the latest stable release of the Flutter SDK:



For other release channels, and older builds, see the [SDK archive](#) page.

2. Extract the zip file and place the contained **flutter** in the desired installation location for the Flutter SDK (for example, `C:\src\flutter`; do not install Flutter in a directory like `C:\Program Files\` that requires elevated privileges).

If you don't want to install a fixed version of the installation bundle, you can skip steps 1 and 2. Instead, get the source code from the [Flutter repo](#) on GitHub, and change branches or tags as needed. For example:

```
C:\src>git clone https://github.com/flutter/flutter.git -b stable
```

You are now ready to run Flutter commands in the Flutter Console.

Vous pouvez également utiliser la **commande git** si vous l'avez installé sur votre ordinateur pour télécharger la dernière **version** du SDK Flutter.

```
git clone https://github.com/flutter/flutter.git -b stable
```

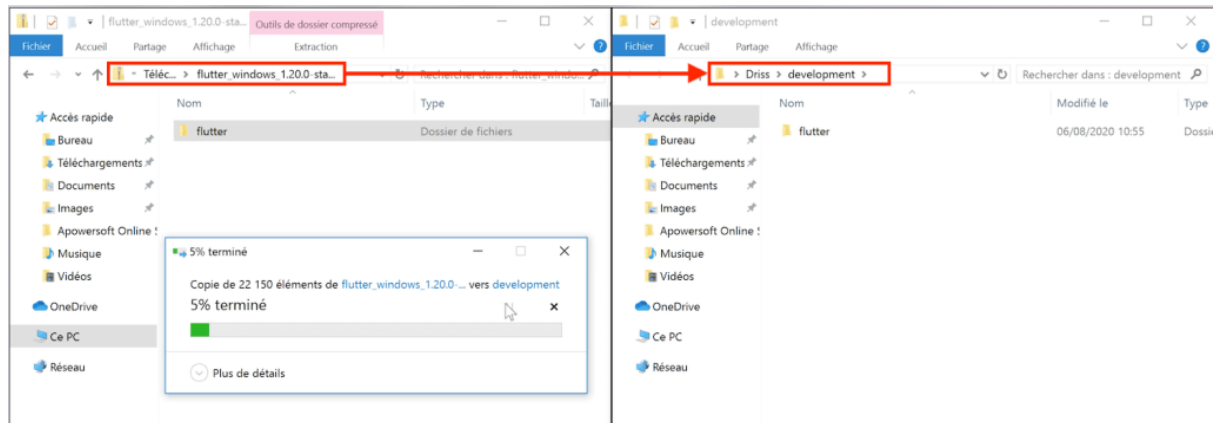
Je vous invite quelle que soit la méthode que vous utilisez, de placer le **SDK Flutter** dans un nouveau **dossier "development"**.

Pour créer ce nouveau dossier à la racine de votre session utilisateur, ouvrez le logiciel **"Console"** ou **"Invite de commande"**.

Une fois ouvert, entrez la commande **mkdir** pour **créer le dossier** development et **start** pour **l'ouvrir** dans votre explorateur de fichier.

```
mkdir development
start development
```

Une fois que ce dossier **development** est ouvert, je vous invite à **transférer le contenu dézippé** du SDK Flutter dans ce nouveau dossier.



Maintenant que votre **SDK Flutter** se trouve dans votre dossier **development**, vous pouvez vous rendre dans le sous dossier **flutter/bin**:

```
cd development/flutter/bin
```

C'est en effet dans ce **sous-dossier** que se trouve notre **commande flutter** qui nous permettra de créer et déployer nos applications.

Vous pouvez par exemple tester la **commande d'analyse "flutter doctor"** qui vous indique si tout est bien installé sur votre ordinateur.

```
flutter doctor
```

Pour le moment, la console devrait vous indiquer **l'absence du SDK Android** sur votre PC.

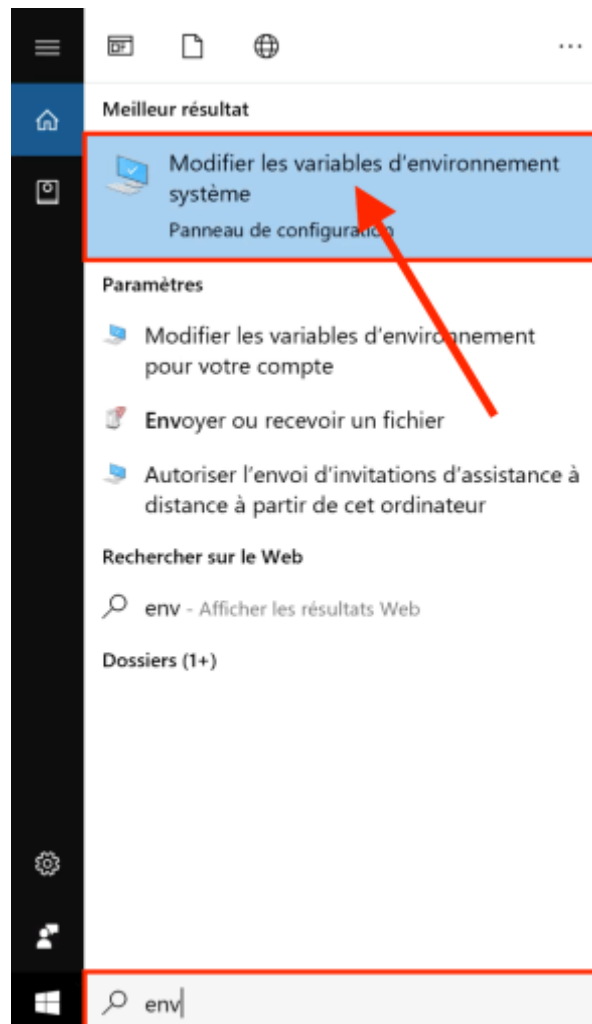
5.2 Configurer le SDK Flutter sur Windows

Nous sommes donc capables d'utiliser la **commande Flutter** et accéder à tout le SDK.

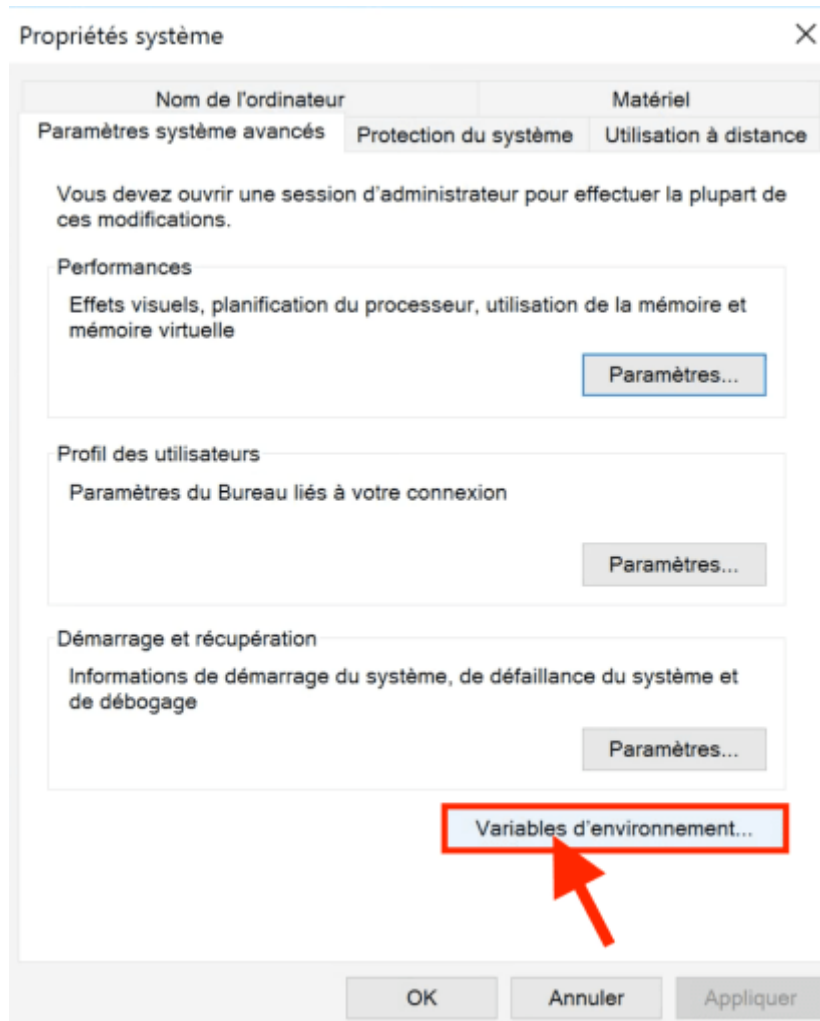
Mais pour l'instant, nous ne pouvons le faire qu'**en nous rendant** dans ce **sous-dossier bin**.

Nous pouvons déclarer et mettre à jour nos **variables d'environnements** pour pouvoir accéder à la commande Flutter depuis **n'importe où**.

Pour cela, tapez dans la barre de recherche Windows “**env**” et cliquez sur le résultat “**Modifier les variables d'environnement**”:



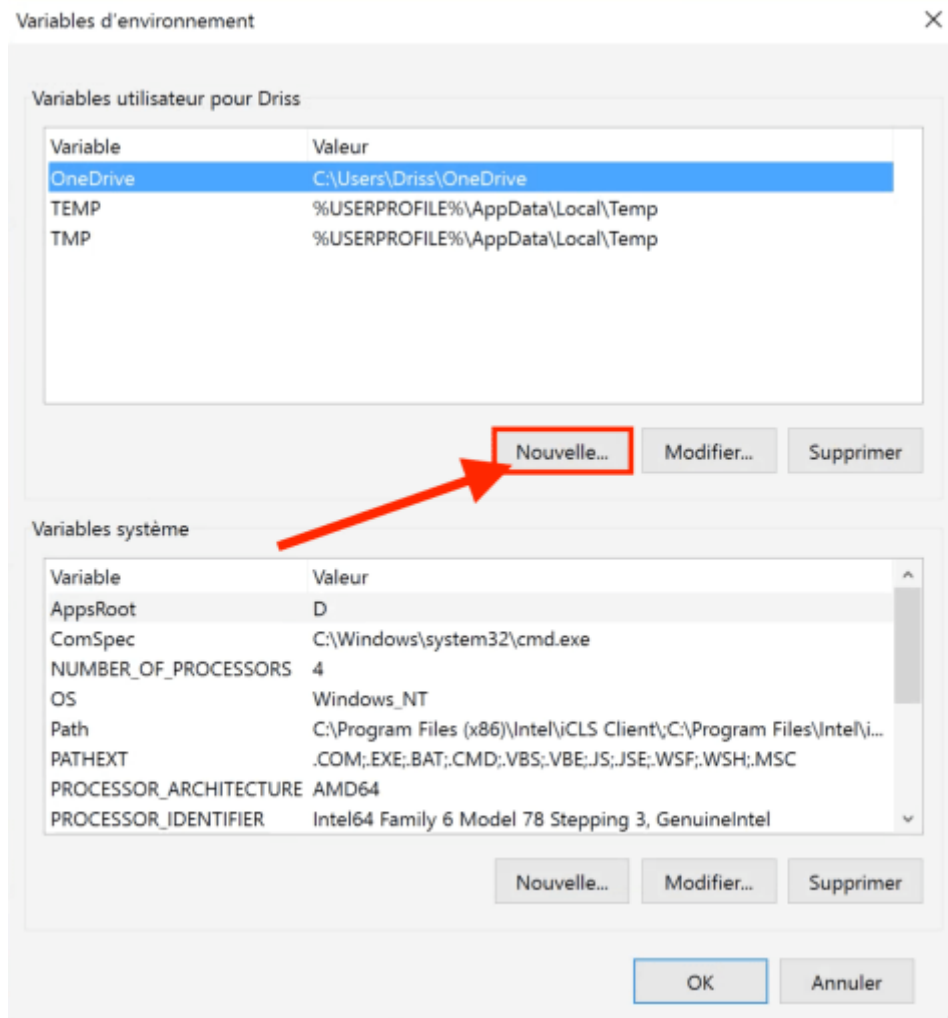
Une fois le **panneau de configuration** ouvert, cliquez sur le bouton “**Variables d'environnement**” en bas de la page:



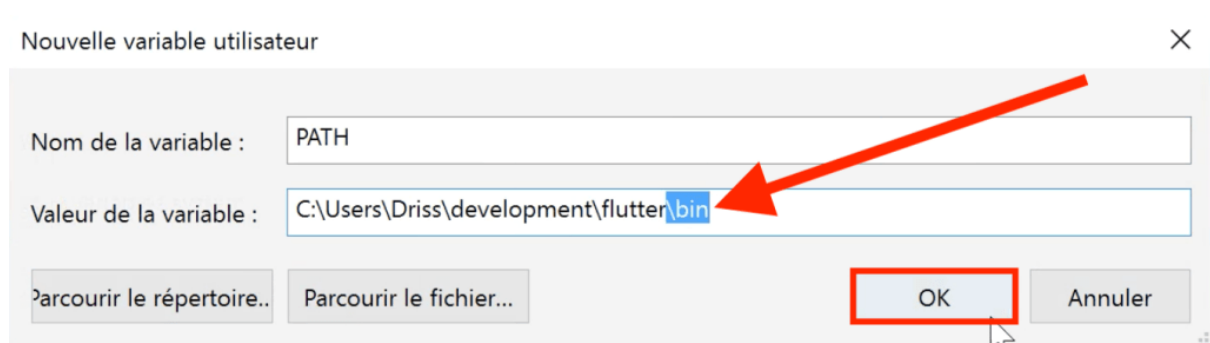
Il devrait alors vous afficher les **variables d'environnement**, déjà présentes sur votre PC.

Si la variable **PATH** ou **Path** existe déjà cliquez dessus et sur **Modifier** pour la mettre à jour.

Si elle **n'apparaît pas encore**, cliquez sur "**Nouvelle**":



Indiquez alors le nouveau de votre variable “**PATH**” et donnez-lui l’adresse de votre **répertoire flutter/bin** de votre disque:



Cliquez ensuite sur **OK** et fermer votre panneau de configuration et **RELANCER** votre terminal.

Lorsque vous entrerez la **commande** suivante **echo** suivante:

```
echo %PATH%
```

Votre console devrait vous **afficher** toutes les **variables d'environnement** de votre PC.

Elles sont séparées par des point virgules et la **dernière** devrait être l'adresse de votre **SDK Flutter flutter/bin**.

Vous avez donc normalement accès à la **commande Flutter** depuis **tout votre PC** pour créer par exemple de nouvelle application.

5.3 Configurer Android Studio

Pour terminer nous allons maintenant **installer et configurer** le logiciel de développement **Android Studio**: <https://developer.android.com/studio>

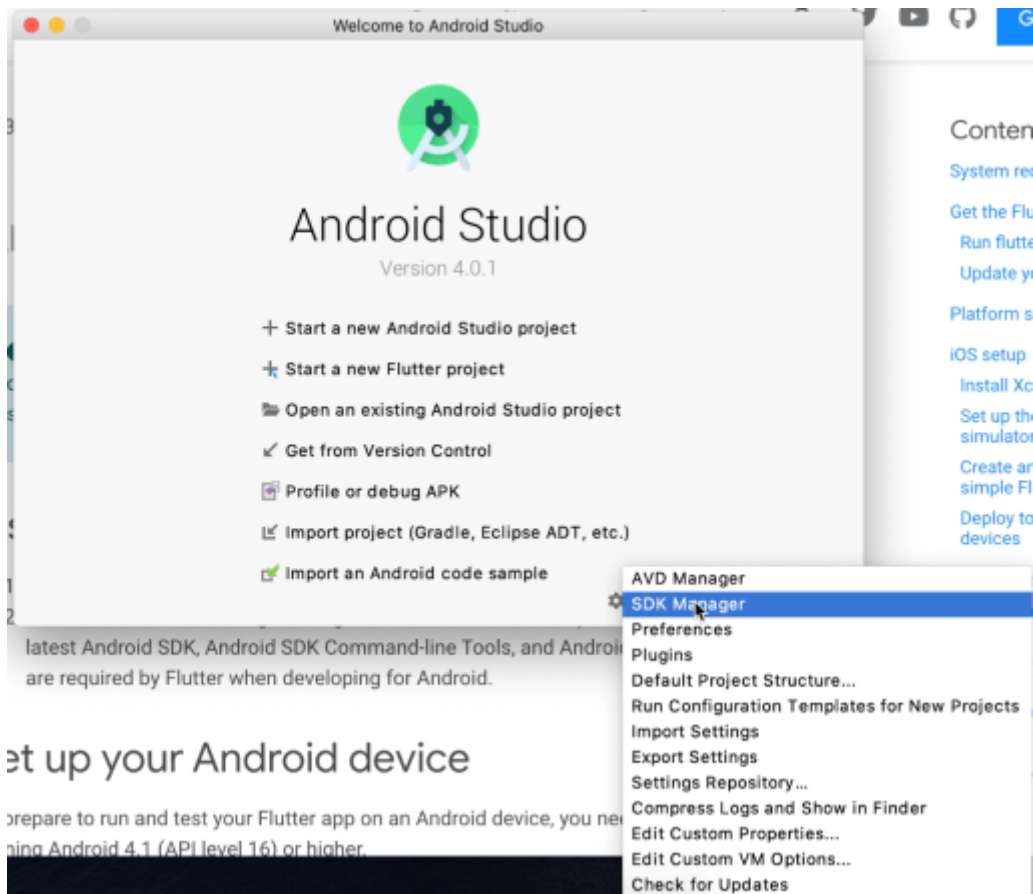


Android Studio provides the fastest tools for building apps on every type of Android device.



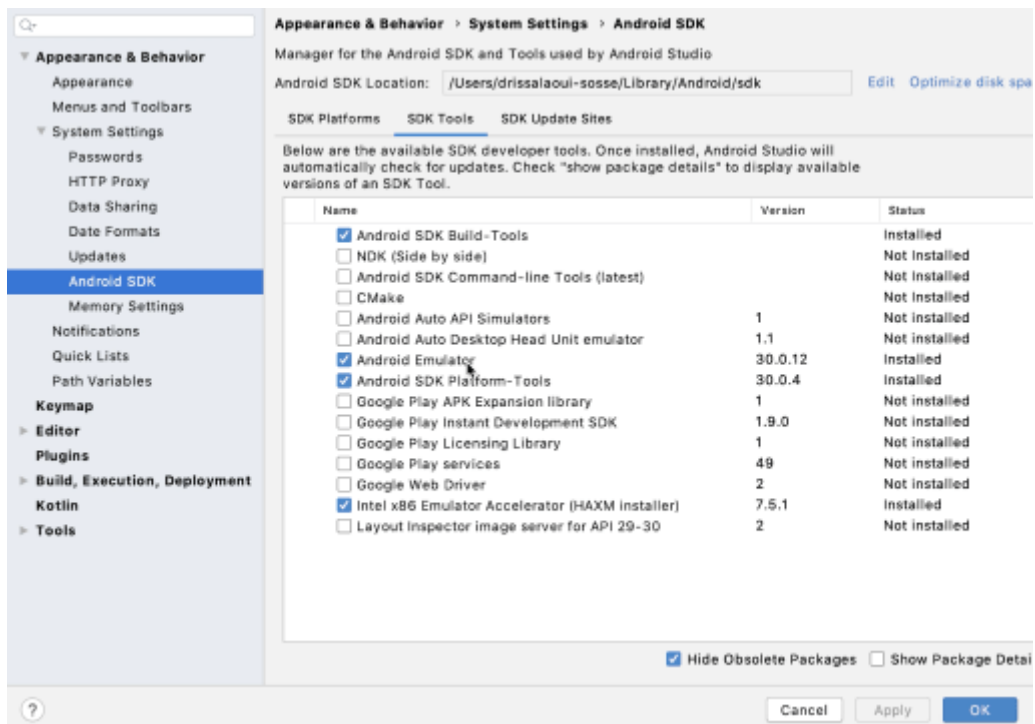
La **procédure** d'installation est la **même** que pour **Mac**, et elle est classique donc pas d'inquiétude à ce niveau-là.

Une fois que le logiciel Android Studio est **téléchargé et installé**, ouvrez-le une première fois.



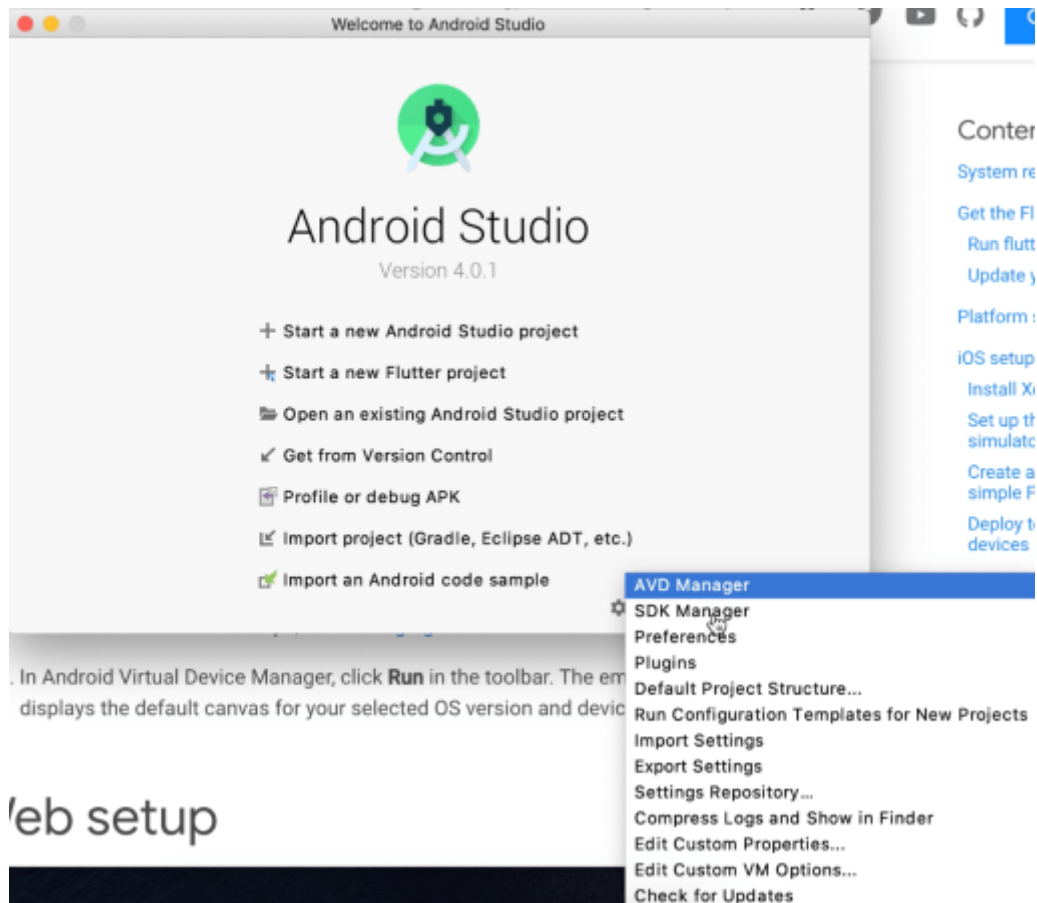
Sur le bouton Configurer, cliquez sur **SDK Manager** pour vérifier la présence de **tous les SDK** dont nous allons avoir besoin par la suite.

Nous utiliserons **Android SDK Build-Tools**, **Android Emulator** et **Android SDK Platform-Tools**.



S'il en **manque un**, cliquez dessus et **installez-le** pour être à jour.

Ensuite revenez sur l'**écran d'accueil** d'Android Studio et cliquez cette fois-ci sur **AVD Manager**:

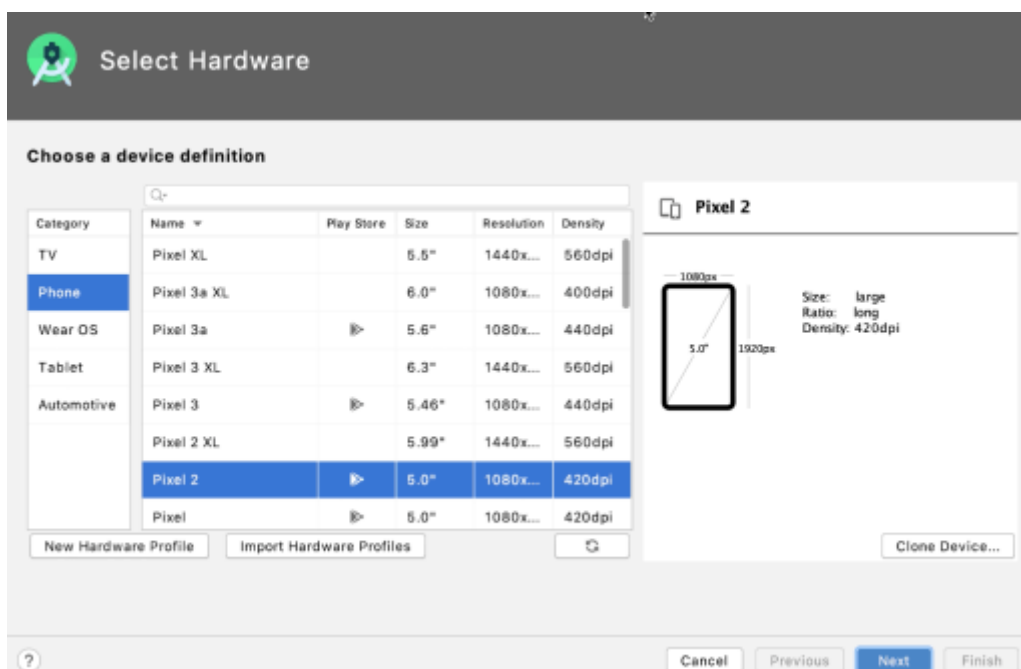


C'est le **gestionnaire des émulateurs Android** (Android Virtual Device: AVD) qui nous permettra de les **lancer** ou d'en **créer** de nouveaux.

Vous pouvez donc cliquer sur “**+ Create Virtual Device**” pour créer notre **premier émulateur** Android:

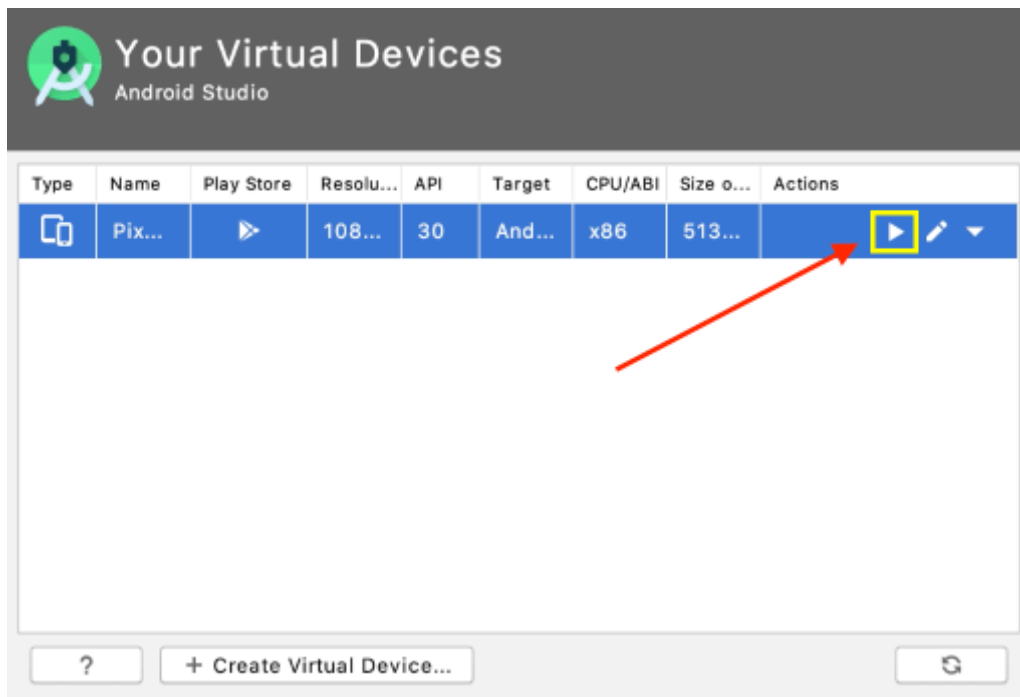


Sélectionnez ensuite le **modèle d'Android** que vous souhaitez reproduire, personnellement je conserve le choix du **Pixel 2** proposé par défaut:



Installez ensuite la **dernière version d'Android** et valider toutes les étapes de la création votre premier émulateur.

Une fois créé, il devrait apparaître dans votre **tableau de bord**, vous n'avez plus qu'à le lancer en cliquant sur l'**icône play**:



Pendant que votre émulateur Android **s'ouvre** et se charge correctement, ouvrez à nouveau votre **console Windows**.

Rendez-vous dans votre **répertoire de développement**:

```
cd development
```

Puis **créez** votre **nouvelle application Flutter** avec la commande:

```
flutter create my_app
```

Un **nouveau dossier** devrait apparaître dans votre répertoire **development**, contenant tout le code de votre **application Flutter**.

Avant de lancer votre application dans l'émulateur Android, exécutez une **dernière fois** la commande:

```
flutter doctor
```

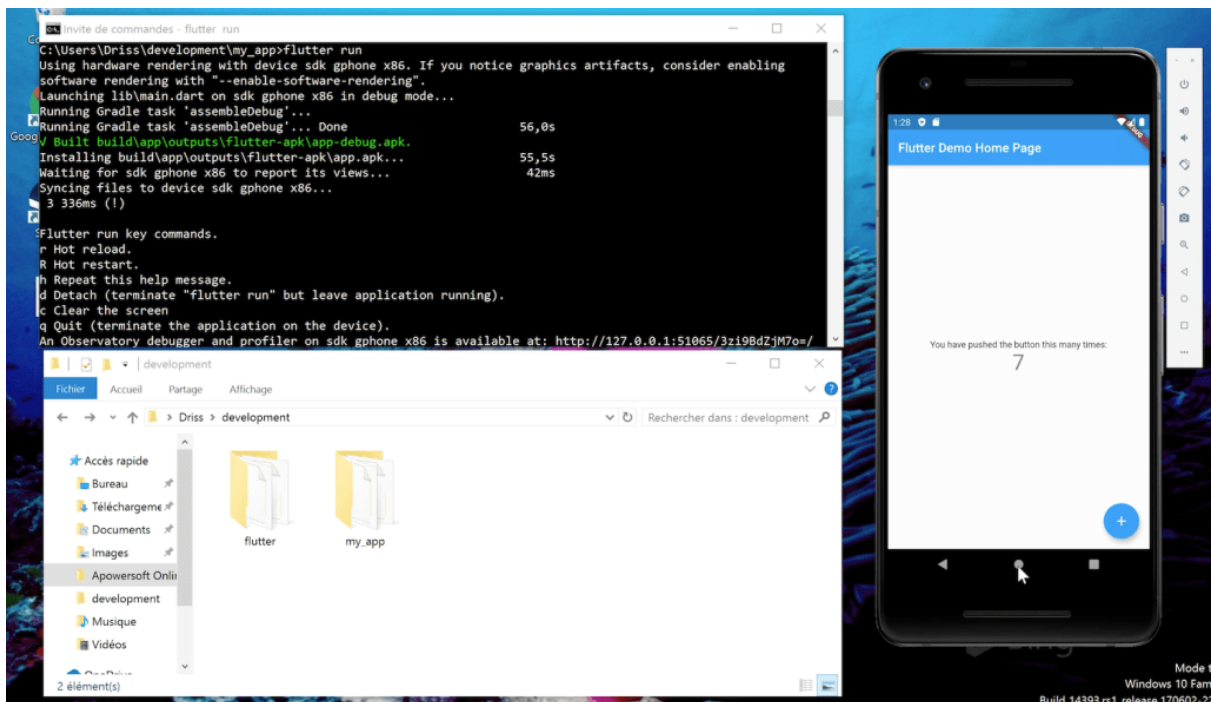
Pour repérer d'éventuelles **erreurs du SDK**, dans mon cas, je devais encore **accepter les conditions d'utilisation** du **SDK d'Android**, j'ai donc dû entrer la commande suivante:

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

Une fois toutes les conditions d'utilisation acceptées, vous pouvez lancer la **commande run** pour lancer votre application:

```
flutter run
```

Patiencez quelques instants pour voir votre application **se lancer** dans votre **émulateur Android**.



Voilà pour l'installation de **Flutter** sur **Windows**.

6. Configurer Visual Studio Code

Dans ce chapitre, nous allons voir comment **installer et configurer** l'éditeur de code **Visual Studio** pour créer nos applications avec Flutter.

Flutter propose en effet **différentes solutions** pour développer avec Dart et Flutter, j'ai choisi d'utiliser Visual Studio Code, car le logiciel est **disponible** sur toutes les plateformes **Mac et PC**.

Nous verrons ensuite comment **créer une nouvelle application Flutter** en partant de zéro et l'ouvrir dans votre émulateur **iOS ou Android** depuis Visual Studio Code (VSC).

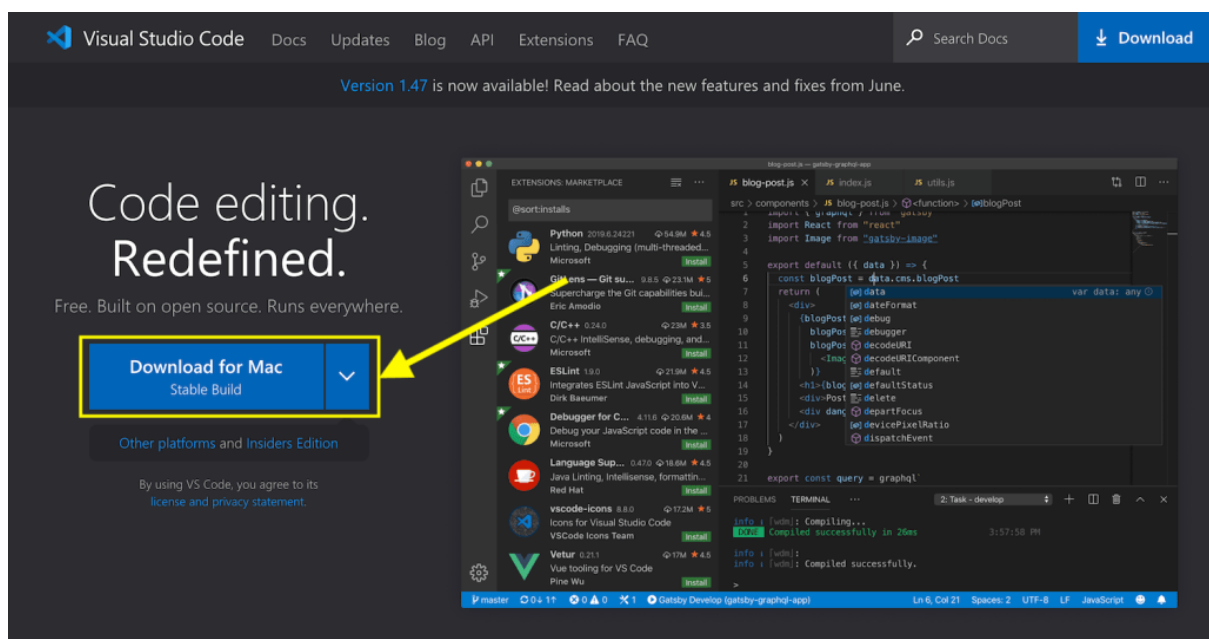
6.1 Installer et configurer Visual Studio Code

Flutter propose une page consacrée à la configuration d'un **éditeur de code**:
<https://flutter.dev/docs/get-started/editor?tab=vscode>

J'ai choisi **VSC** car Flutter y a créé des **extensions** qui transforment l'éditeur en véritable **logiciel** de développement très **complet**.

Nous n'aurons même **plus besoin d'ouvrir Xcode ou Android Studio** pour lancer nos applications dans un émulateur.

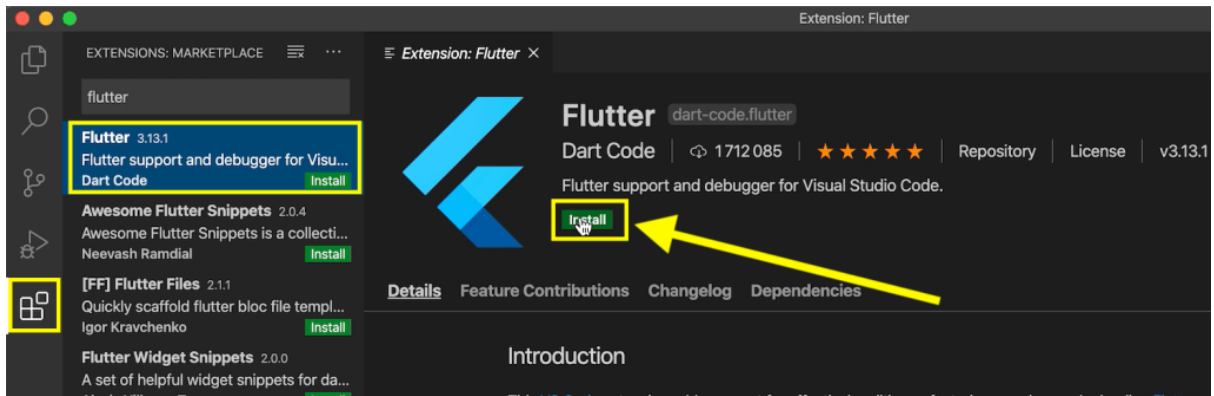
Vous pouvez commencer par **télécharger le logiciel VSC** depuis le site:
<https://code.visualstudio.com/>



Une fois le logiciel installé et lancer sur votre ordinateur, nous allons installer les **plugins Dart et Flutter** de VSC.

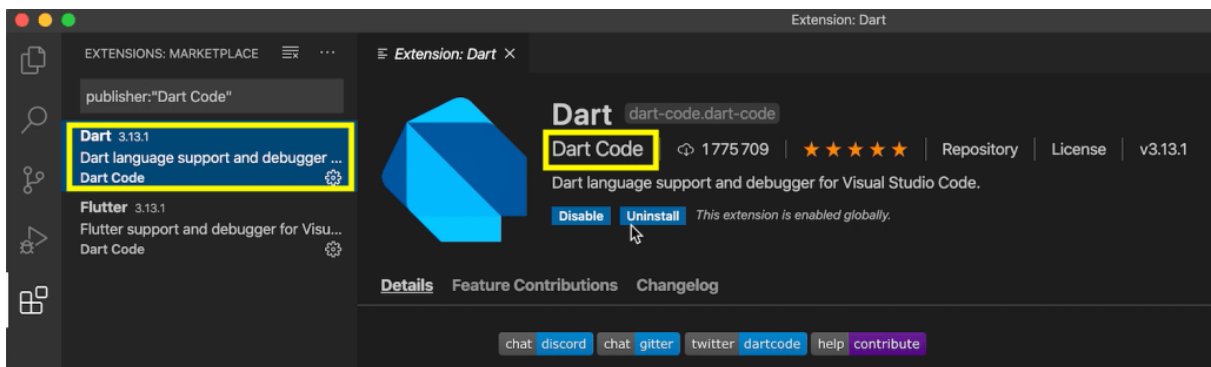
Ouvrez le **menu des extensions** à gauche de votre logiciel, puis tapez dans la **barre de recherche** "Flutter".

Installer ensuite le premier **plugin** qui s'affiche qui doit avoir été **édité par Dart Code**.



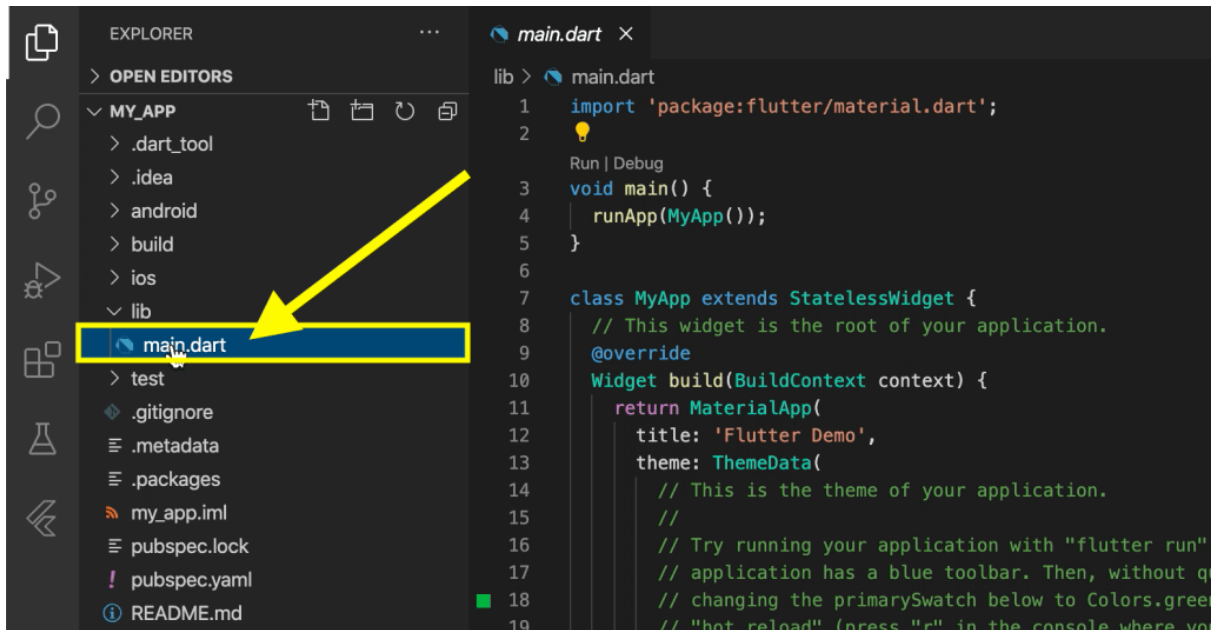
Lorsque le **plugin Flutter** s'installe, le **plugin Dart** doit lui aussi être **installé automatiquement**.

Dans le doute, vérifiez qu'il est bien installé en cherchant le **plugin Dart** du développeur Dart Code.



Maintenant que les **deux extensions** sont **installées**, nous allons **tester la fonctionnalité** la plus intéressante de VSC.

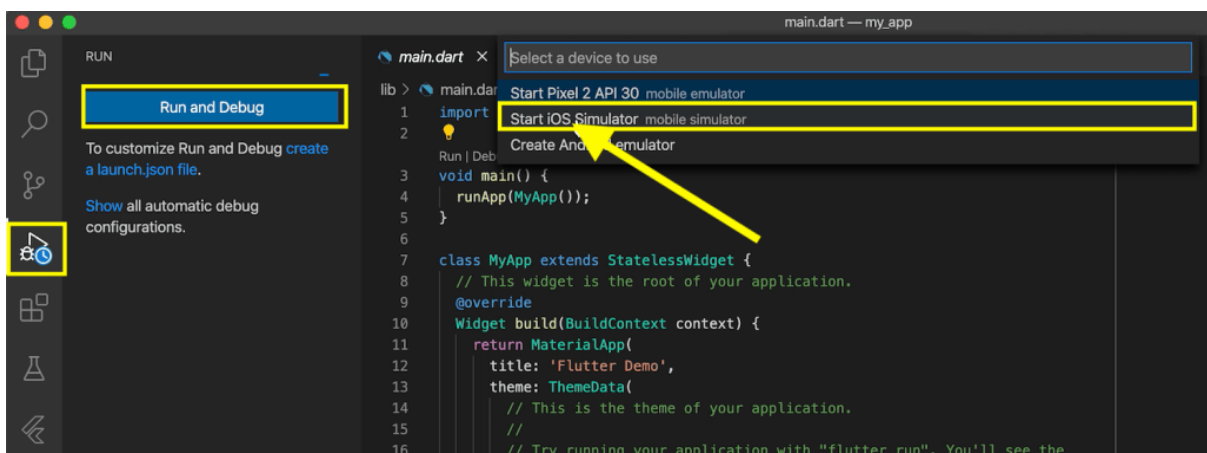
Commencez par **ouvrir le fichier principal** de votre application, le fichier ***main.dart*** dans le dossier ***lib***:



Maintenant que ce **fichier est ouvert**, c'est celui-ci que VSC choisira **d'exécuter et de lancer** dans votre émulateur.

Rendez-vous ensuite dans l'onglet Run de votre **menu latéral** et cliquez sur **"Run and Debug"**.

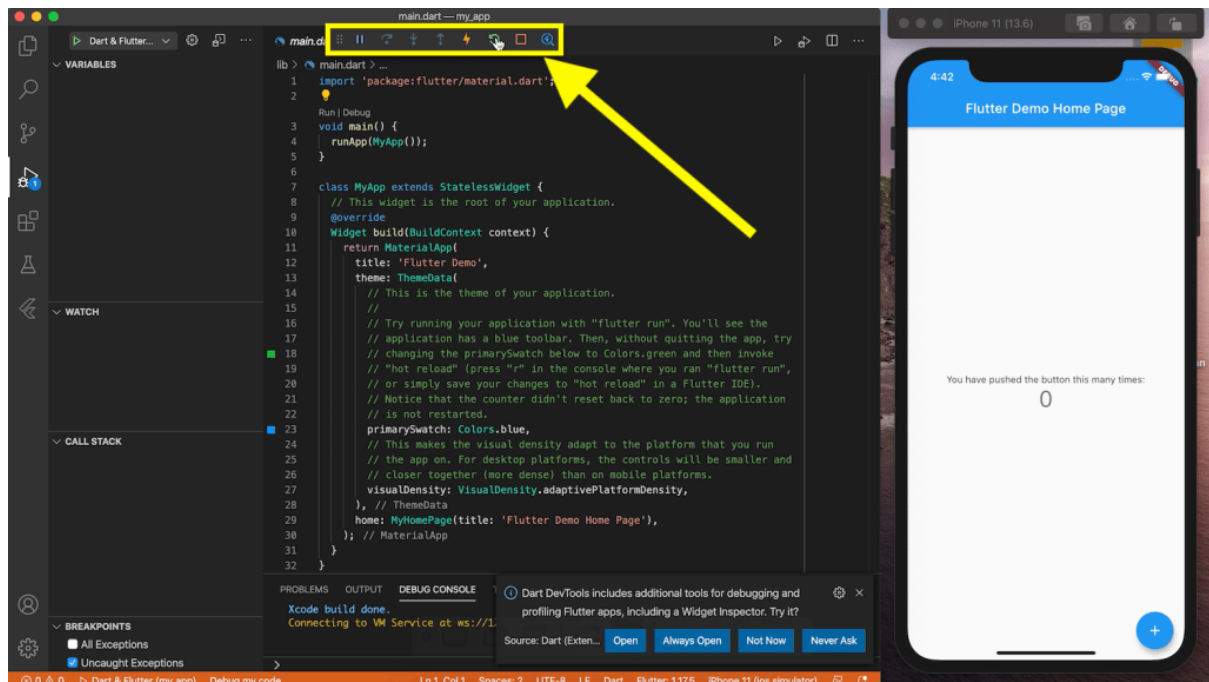
VSC devrait vous proposer de **choisir un émulateur iOS ou Android** selon votre configuration.



Je choisis de lancer mon application sur un **émulateur iOS** et il me lance mon dernier appareil ouvert, un **iPhone 11**.

Après un **petit temps de chargement** mon application s'ouvre dans mon émulateur.

Vous avez accès à une **série de boutons** pour **rafraîchir** manuellement votre application ou bien **d'arrêter** son exécution.



Mais normalement dès que vous **enregistrez votre fichier** main.dart, votre application devrait **s'actualiser en direct**, voilà toute la magie de Flutter!

6.2 Créer une application en partant de zéro

Je vous montre à nouveau comment **créer une application Flutter** en partant de zéro.

L'idée ici est de vous montrer la **procédure à suivre** lorsqu'on utilise **VSC** pour coder ses applications Flutter.

Vous devez comme toujours ouvrir votre **terminal** et vous rendre dans le dossier “**development**” qui contient votre SDK Flutter et vos autres applications.

```
cd development
```

Une fois que vous vous trouvez dans ce **dossier**, vous pouvez entrer la commande “**flutter create**” pour créer une nouvelle application Flutter avec son **nom**:

```
flutter create my_flutter_app
```

Cette étape doit être effectuée depuis votre logiciel **Terminal ou Console**, et non dans Visual Studio.

Vous avez le choix à partir de d'**ouvrir votre nouvelle application dans Visual Studio** et de suivre les procédures précédentes pour lancer votre application dans un émulateur.

Sinon vous pouvez aussi **rester dans votre terminal**, vous rendre dans le dossier de votre application et la lancer avec la commande "**flutter run**":

```
cd my_flutter_app  
flutter run
```

Sinon je vous conseille de n'utiliser le **Terminal ou la Console** que pour **créer de nouvelles applications**, ce qui ne devrait pas arriver si souvent.

Ensuite je vous invite à **travailler exclusivement avec Visual Studio** qui intègre son **propre terminal**.

Cela vous **facilitera votre travail** de développeur en ayant à ouvrir **que le logiciel Visual Studio** lorsque vous voulez améliorer votre application.