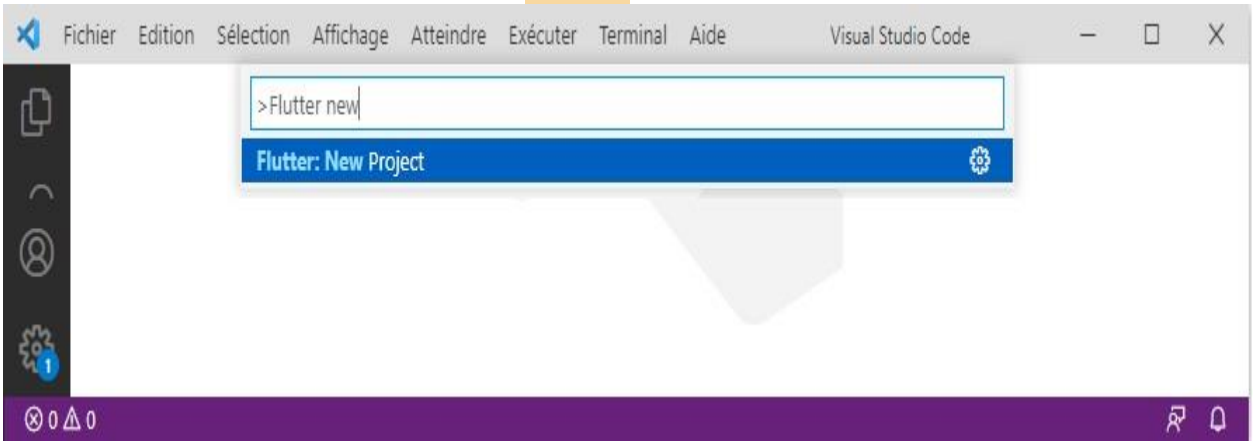
Créez votre première application Flutter

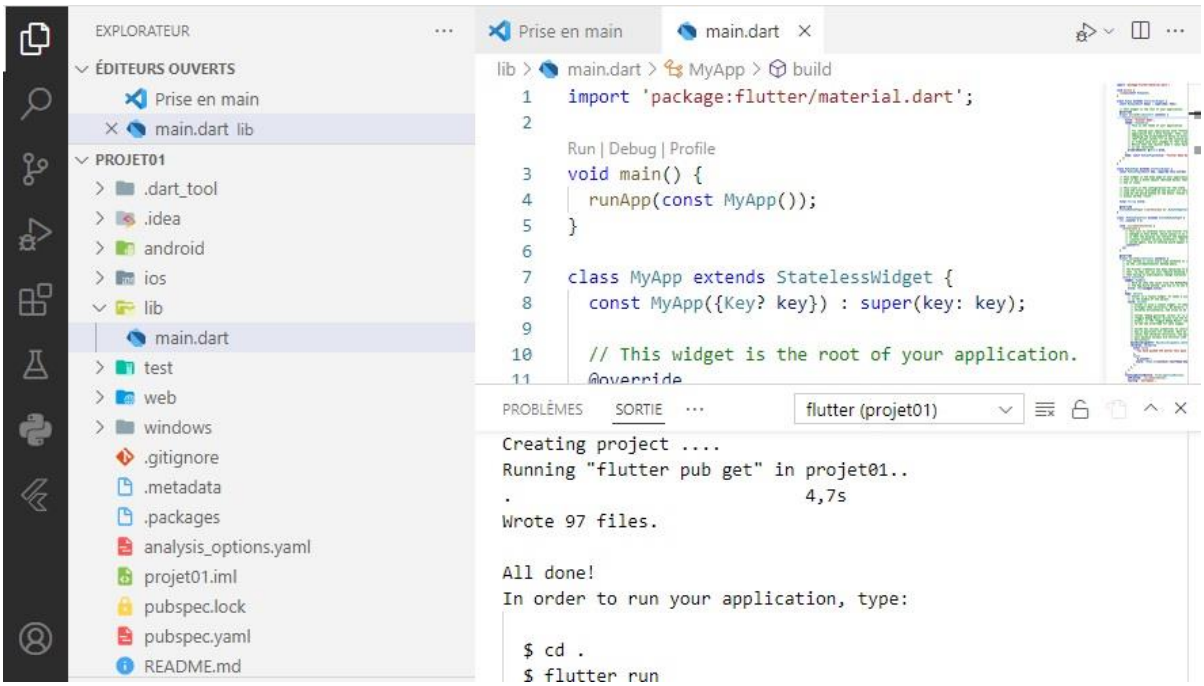
1. Création du projet Flutter avec Visual Studio Code

- Commençons par ouvrir Visual Studio Code. Vous arrivez sur la page d'accueil.
- Vous arrivez ensuite sur une fenêtre qui vous indique les différents raccourcis pour les différentes opérations qui nous intéressent (ouvrir un fichier etc.).

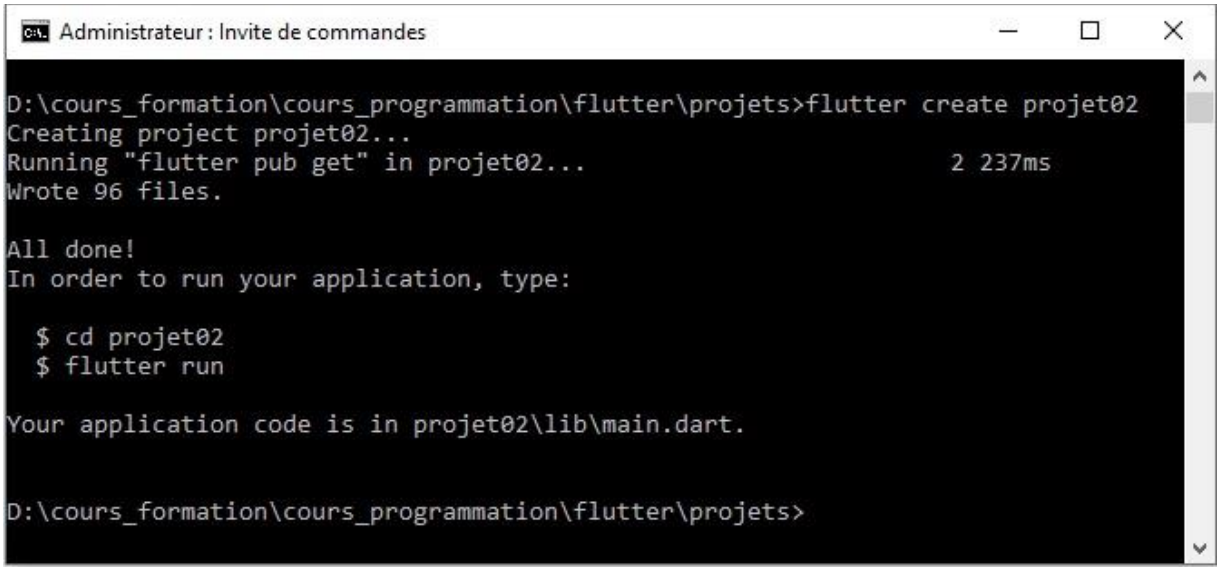
- **Méthode 01**
  - Ensuite affichez la barre d'accès avec la combinaison de touches "Ctrl + Shift + P" sur Windows.
  - Dans la barre de recherche tapez "**Flutter new**" et sélectionnez ensuite "**Flutter : New Application Project**" puis pressez "`Entrer`".



  - Vous êtes ensuite invité à entrer le nom de votre application. Choisissez le bien car tous les fichiers seront créés en référence à ce nom bien qu'il ne détermine pas le nom de votre application telle qu'elle apparaîtra une fois publiée.
  - Entrez un le nom de votre application. Si le nom de votre application est composé de plusieurs mots, insérez un _ entre les différents mots comme ceci par exemple: mon_app_révolutionnaire. Notez que vous ne pouvez pas utiliser de majuscules pour le nom de votre application.
  - Vous obtenez ainsi un projet comme ci-dessous :
  - Visual Studio Code va créer tout un lot de fichiers sans que vous ayez à cliquer ou taper quoi que ce soit.



- **Méthode 02**
  - Vous pouvez aussi lancer un nouveau projet sans utiliser Visual Studio Code.
  - Pour cela, il faut lancer le terminal de commande, se mettre à l'emplacement du futur projet Flutter puis de lancer la commande **"flutter create"** suivie du nom de votre projet comme dans l'exemple ci-dessous.

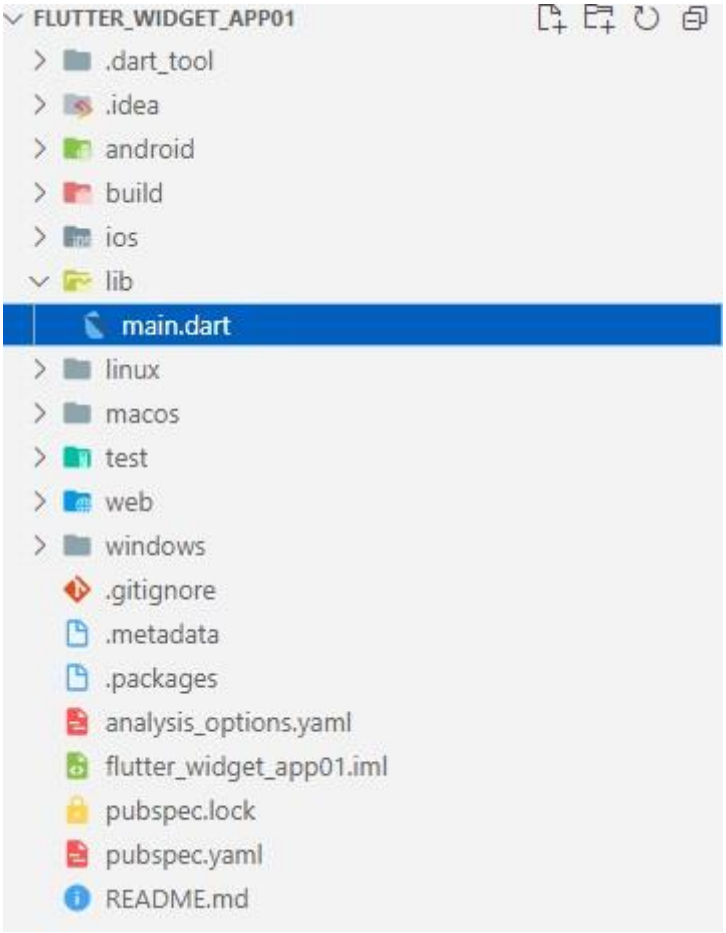


- Cette commande va générer les lignes de code de votre projet. Ensuite il vous suffira d'aller chercher le projet crée à son emplacement et de l'ouvrir avec **Android Studio** ou **VS Code**.

2. Structure du projet

Un projet Flutter regroupe deux éléments principaux :

- Le dossier du projet (ici, WIDGET_FLUTTER_APP01) où tous les fichiers manipulables sont stockés. C'est la partie principale.
- Les bibliothèques externes (External Libraries), dossier qui contient les éléments utiles au bon fonctionnement du projet tels que les packages et SDK de Dart.



- **`android`**:
- Le dossier génère automatiquement le code pour l'application d'Android.
- Il contient tous les fichiers spécifiques à Android sont contenus ici. Il est possible de retrouver à l'intérieur les fichiers :

✓ **`gradle`** : les fichiers Gradle sont des fichiers de configuration. De nombreuses informations sont contenues ici comme la version de l'API Android (Application Programming Interface ou interface de programmation d'application en français). La version de l'application y est également présente. Enfin, les bibliothèques utiles à l'application y sont déclarées.

✓ **`app`** : Ce dossier contient le code de l'application Android. Dans le dossier src puis main est situé le fichier AndroidManifest.xml. C'est le fichier auquel il faudra recourir pour attribuer des permissions spéciales à certaines fonctionnalités comme l'accès à Internet ou encore l'envoi de SMS.

- **`ios`** : .
- Le dossier génère automatiquement le code pour l'application d'iOS
- Tous les fichiers relatifs à iOS sont rangés à cet endroit.
- **`lib`** :
- Le dossier d'accueil contient le code Dart de l'application.

- il s'agit du dossier qui va nous intéresser dans un premier temps. En effet, le fichier `main.dart` s'y trouve. Le code présenté dans la zone centrale au démarrage est celui de ce fichier. C'est le cœur de l'application. Plus tard, il sera possible de le découper afin de lui laisser uniquement la charge du lancement de l'application. Le reste du code, lui, sera dispersé logiquement dans d'autres fichiers.

✓ **`lib/main.dart`** : Le fichier est convoqué pour démarrer (start) l'application.

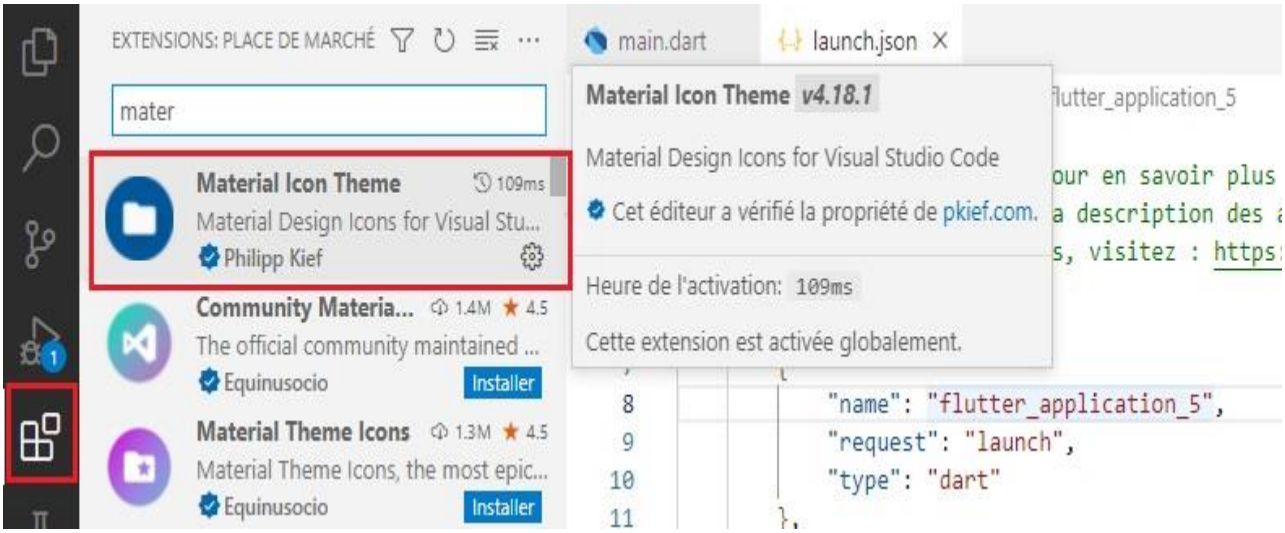
- **`test`** :
- ce dossier contient le fichier Dart widget_test.dart qui, comme son nom l'indique, est dédié aux tests. C'est dans ce dossier que l'ensemble du code servant à tester l'application devra se situer.
- Le dossier contient les codes Dart pour tester l'application.

✓ **`test/widget_test.dart`** : Exemple de code

- **`pubspec.yaml`** :
- Un fichier utilisé pour déclarer les ressources relatives au projet comme images, polices, etc.
- Il s'agit d'un fichier de déclarations. Ici, les ressources (assets) telles que des images sont déclarées, tout comme les bibliothèques externes utiles au projet. Il s'agit d'un fichier majeur auquel il faudra recourir régulièrement.
- **`.gitignore`** : Git version control file – Ce dossier contient la configuration du projet GIT.
- **`.metadata`** : Le dossier est automatiquement généré par l'outil de Flutter.
- **`.packages`** : Automatiquement généré, le fichier contient une liste de dépendances utilisées par le projet.
- **`.iml`** : Un fichier de projet d'Android Studio.
- **`pubspec.lock`** : Ce fichier doit être ajouté à GIT Control pour s'assurer que les membres de votre équipe de développement utilisent les mêmes versions de bibliothèque.
- **`README.md`** : Le fichier décrit le projet, lequel est écrit selon la structure Markdown.

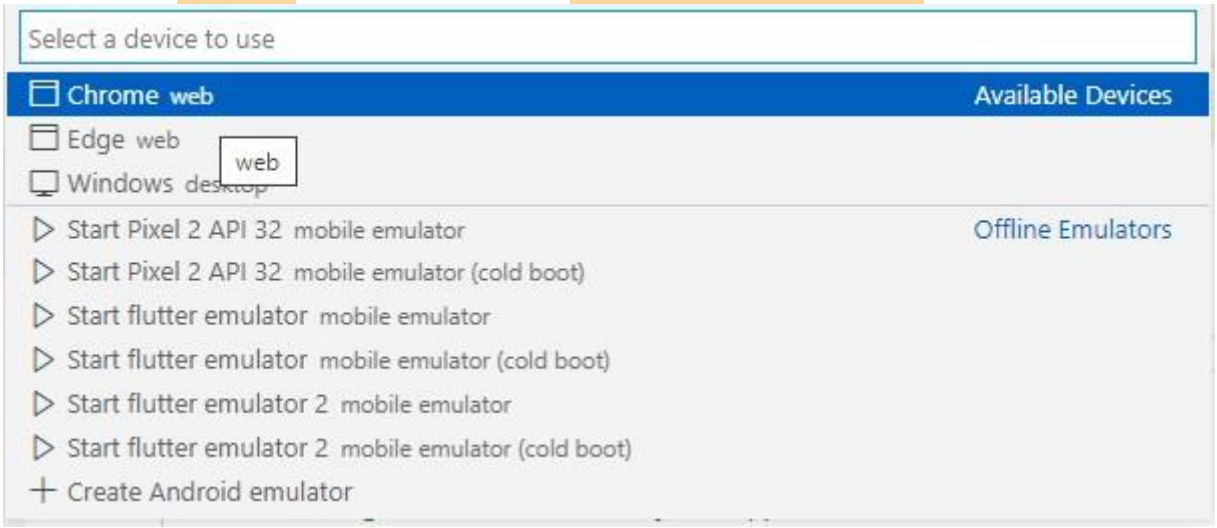
3. Installer quelques icônes

- Revenons à notre projet, avant de se lancer dans le code, on va installer quelques icônes. Cliquez sur le symbole encadré en blanc dans la barre de gauche puis dans la barre de recherche tapez "`material icon theme`".
- Installer "`Material Icon Theme`" vous permettra d'avoir une meilleure visibilité de vos dossiers.

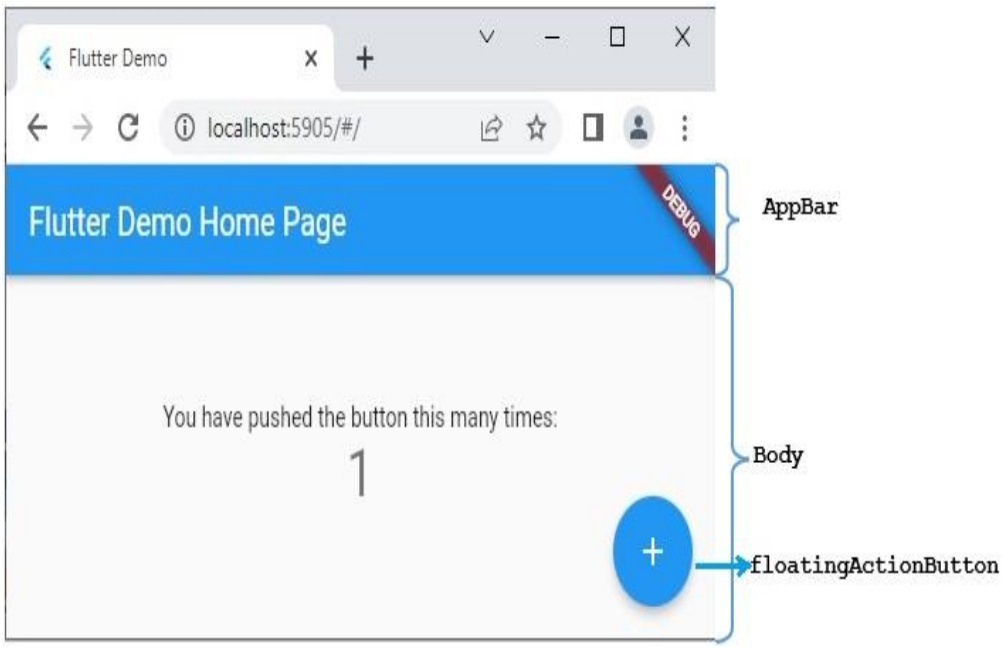


4. Lancer l'application

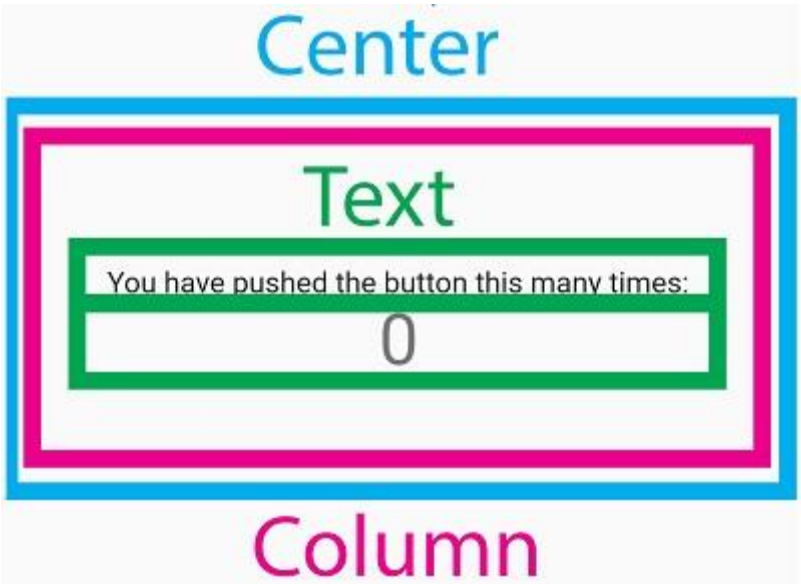
- Avant de lancer l'application choisissez l'émulateur. Pour ce faire, tapez dans la barre de commande " `Flutter` " puis sélectionnez " `Flutter : Select Device` ".



- Pour lancer le debugging et la compilation appuyez sur F5 ou la combinaison des touches "Fn" + "F5" ou sinon aller dans le menu, cliquer sur Debug > Start Debugging.
- Comme c'est la première fois qu'on lance l'application, ça va prendre plus de temps.
- Pour mon cas je vais choisir Chrome
- Une fois lancé le résultat sera l'écran ci-desssous



- L'application de base générée par la commande Flutter : New project n'est pas l'habituel hello world mais une application contenant un texte "You have pushed the button this many times: N" et un bouton. Le clic du bouton incrémente le nombre affiché dans le texte.
- Le widget du body est ici composé d'un widget **Center**, lui même composé d'un widget **Column**, lui même composé deux deux widgets **Text**.



- **Application**
- Changer le nom de l'application qui s'affiche en haut du téléphone : "Flutter Demo Home Page".

5. Analyser la composition du fichier main.dart.

main.dart

lib > main.dart > MyApp > build

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

Run | Debug | Profile
void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        // This is the theme of your application.
        //
        // Try running your application with "flutter run". You'll see the
        // application has a blue toolbar. Then, without quitting the app, try
        // changing the primarySwatch below to Colors.green and then invoke
        // "hot reload" (press "r" in the console where you ran "flutter run",
        // or simply save your changes to "hot reload" in a Flutter IDE).
        // Notice that the counter didn't reset back to zero; the application
        // is not restarted.
        primarySwatch: Colors.blue,
      ), // ThemeData
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    ); // MaterialApp
  }
}
```

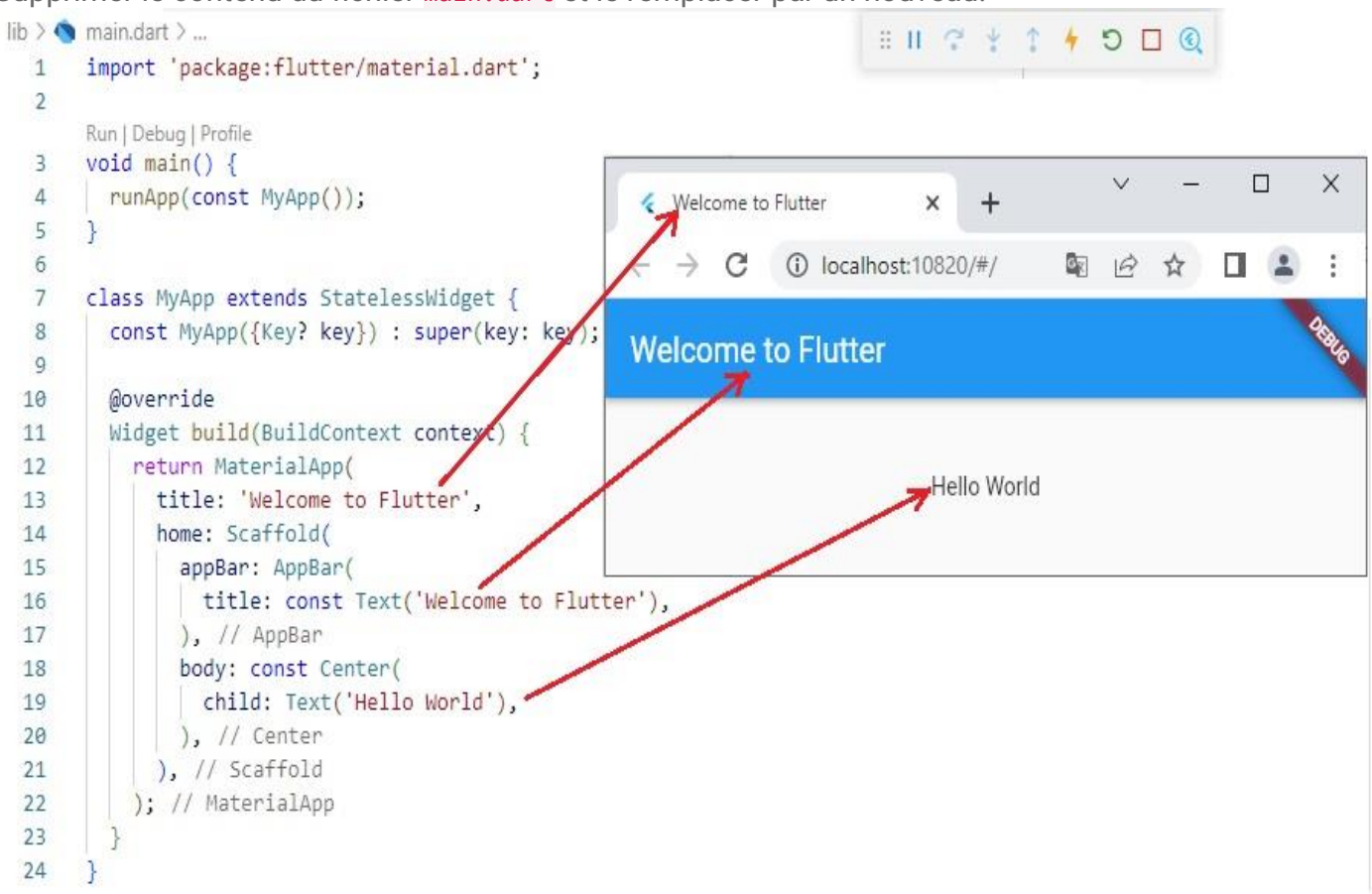
- La ligne :

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

- Elle est indispensable à tout projet car elle contient les ressources nécessaires à l'application comme les Widgets.
- La ligne :

```
void main() => runApp(MyApp());
```

- Elle représente la porte d'entrée principale de l'application, c'est grâce à cette ligne qu'on lance le code de l'application.
- Flutter est constitué essentiellement de Widgets. Un widget c'est comme un Lego, vous devez imbriquer plusieurs widgets ensemble pour créer votre application.
- **Changer le contenu du fichier main.dart**
- Supprimer le contenu du fichier `main.dart` et le remplacer par un nouveau.



- Cet exemple crée une application Material. Material est un langage de conception visuelle standard sur mobile et sur le Web. Flutter propose un riche ensemble de widgets Material.
- L'application s'étend StatelessWidget, ce qui fait de l'application elle-même un widget. Dans Flutter, presque tout est un widget, y compris l'alignement, le remplissage et la mise en page.

- Le Scaffoldwidget, de la bibliothèque de matériaux, fournit une barre d'application par défaut, un titre et une propriété body qui contient l'arborescence du widget pour l'écran d'accueil. La sous-arborescence des widgets peut être assez complexe.
- Le travail principal d'un widget est de fournir une buildméthode qui décrit comment afficher le widget en termes d'autres widgets de niveau inférieur.
- Le corps de cet exemple consiste en un **Centerwidget** contenant un **Textwidget** enfant. Le **Centerwidget** aligne sa sous-arborescence de widgets au centre de l'écran.