

Programmation Multiplateforme en Flutter

TP 2 : Création d'une première application

Notions à voir :

- Création projet flutter et Structure d'un projet Flutter.
- Première application Flutter, et quelques widgets de base :

MaterialApp.

Scaffold.

AppBar.

Boutons.

Text.

Image.

Rappel :

Une application Flutter est une application où le développeur construit une interface graphique en ajoutant des widgets. Les widgets sont des composants, les blocs constructifs de votre interface utilisateur :

- Un bouton est un widget
- Du texte est un widget
- La barre est un widget
- Tout est widget...

I. Création d'un projet Flutter

Vous pouvez créer l'application en se basant sur l'interface graphique de l'IDE (Android Studio/VS Code...) ou à partir de la ligne de commande :

1. Ligne de commande :

- Étape 1 : **flutter create projet_flutter**.

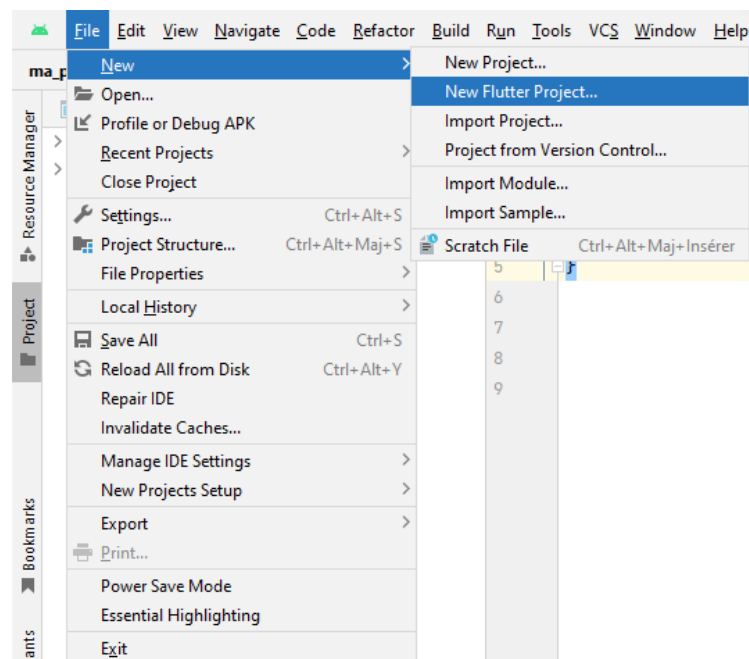
Cette commande flutter vous permet de créer le dossier de votre projet qui contient les fichiers initiaux de votre projet.

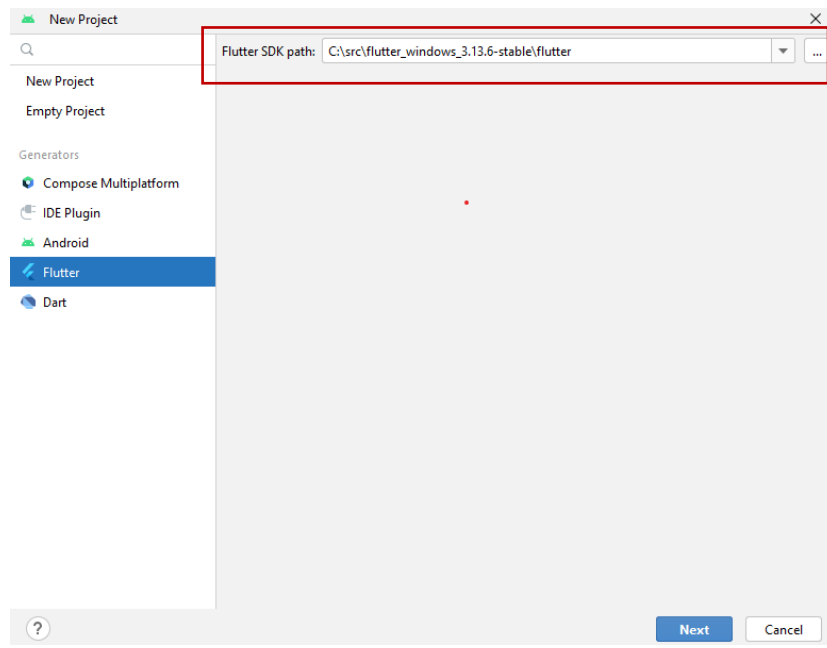
- Étape 2 : **flutter run lib/main.dart**

Le nom du projet flutter ne doit pas contenir les symboles « - » ou « . » etc. Par convention, le nom du projet flutter doit être en minuscule pouvant contenir des underscore « _ ».

2. Environnement de développement intégré (IDE) :

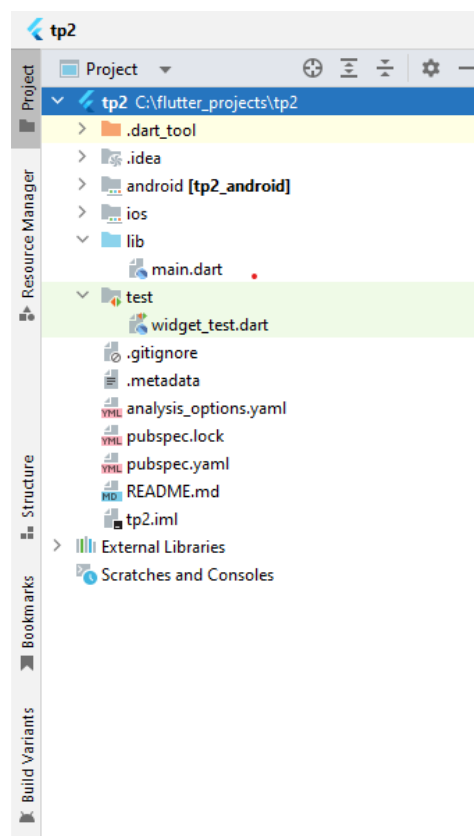
- Création de projet Dans le menu fichier, cliquez sur **nouveau -> projet -> flutter**. Ensuite choisissez l'emplacement de votre flutter SDK.





- Nommez votre projet comme vous voulez, évitez les Maj et les caractères spéciaux.
L'architecture de votre projet va être présentée comme suit :


II. Structure d'un projet Flutter



- Un projet Flutter se compose de divers dossiers :
 - **Android** : Le dossier génère automatiquement le code pour l'application d'Android.
 - **Ios** : Le dossier génère automatiquement le code pour l'application d'iOS.
 - **Lib** : Le dossier d'accueil contient le code Dart de l'application.
 - **lib/main.dart** : Le fichier est convoqué pour démarrer (start) l'application.
 - **Test** : Le dossier contient les codes Dart pour tester l'application.
 - **test/widget_test.dart**: Sample code.
 - **.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.yaml** : Un fichier utilisé pour déclarer les ressources relatives au projet comme images, polices, etc.
 - **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.
- Exécutez l'application générée par défaut dans un émulateur de votre choix afin de comprendre le fonctionnement de base.

III. Création et manipulation de quelques widgets de base :

- Supprimez tout le contenu du « main.dart » puis exécutez le code suivant :



```

1
2
3 import 'package:flutter/material.dart';
4
5
6 >> void main() {
7     runApp(Text('Bonjour'));
8 }
9

```

- Que-ce que cela génère ? Et pourquoi ?

Note :

- Pour créer un Widget, il faut créer une classe qui hérite d'une classe prédéfinie fournie par Flutter (soit dans le cadre de cet exemple la classe « StatelessWidget »).
- Pour faire appel à cette classe, il faut importer le package import « 'package:flutter/material.dart' ».
- Par convention, le nom de la classe ne doit pas contenir d'underscore ou de symbole, et il est préférable que son nom commence par une majuscule.
- Pour hériter de la classe de L'API « StatelessWidget », il faut redéfinir la méthode « build() » de cette classe.
- Cette méthode build() doit avoir comme paramètre l'objet « BuildContext » (cet objet est également fourni par le package importé).
- La méthode « build(BuildContext context) » doit retourner un objet de type Widget : Widget « build(BuildContext context) ».
- Sur la méthode « main() », on a besoin d'exécuter un code pour faire appel à la méthode « build » du widget créé « MyApp » ; qui va nous permettre de dessiner le widget sur l'écran.

- Remplacer le code précédent par celui-là, puis exécutez :

```
main.dart x
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(MyApp());
5 }
6
7 class MyApp extends StatelessWidget{
8   // La méthode build retourne un widget
9   Widget build(BuildContext context){
10    // La méthode MaterialApp() est fournie par le package material.dart
11    // utilise un argument nommé
12    return MaterialApp(home : Text('Bonjour à tous !'),) ;
13  }
14 }
```

- Qu'est-ce que cela affiche ?
- Exécutez le code ci-dessous.

```
main.dart x
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(MaterialApp(home: Scaffold(
5     appBar: AppBar(title: Text(' Ma Première Application Flutter')),
6     body: Text(' Bienvenue', ),),)); // Scaffold, MaterialApp
7 }
```

- Qu'est-ce que vous remarquez au niveau du code ?
- Ajoutez les widgets « AppBar » et « Text » en exécutant le code suivant :

```
main.dart x
2
3 import 'package:flutter/material.dart';
4
5
6 void main() {
7   runApp(MyApp());
8 }
9
10 class MyApp extends StatelessWidget {
11   Widget build(BuildContext context) {
12     return MaterialApp(home: Scaffold(
13       appBar: AppBar(title: Text(' Ma Première Application Flutter')),
14       body: Text(' Bienvenue', ),),); // Scaffold, MaterialApp
15   }
16 }
```

- Faites les manipulations suivantes sur le code précédent pour changer la couleur du texte dans « l'AppBar » et centrer le texte dans le « Scaffold » :

```

1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(MyApp());
5 }
6
7 class MyApp extends StatelessWidget {
8   Widget build(BuildContext context) {
9     return MaterialApp(home: Scaffold(
10      appBar: AppBar(title: Text(' Ma Première Application Flutter',
11        style: TextStyle(color: Colors.white70)), // Text, AppBar
12      body: Center(
13        child: Text(' Bienvenue',
14          style: TextStyle(
15            fontSize: 60,
16            color: Color.fromRGB(100, 10, 10, 100)), // TextStyle, Text, Center, Scaffold, Mate
17      )
18    )
19  }

```

- Utilisez les Widgets de mise en forme à savoir « Column » et « Center » en exécutant le code suivant :

```

7 class MyApp extends StatelessWidget {
8   @override
9   Widget build(BuildContext context) {
10    return MaterialApp(
11      home: Scaffold(
12        appBar: AppBar(title: Text('Exemple de centrage espacé de boutons dans une colonne')),
13        body: Center(
14          child: Column(
15            mainAxisAlignment: MainAxisAlignment.center,
16            mainAxisAlignment: MainAxisAlignment.min, // Pour espacer les boutons
17            children: <Widget>[
18              ElevatedButton(
19                onPressed: null,
20                child: Text('Bouton 1'), // ElevatedButton
21                SizedBox(height: 16), // Espacement entre les boutons
22              ElevatedButton(
23                onPressed: null,
24                child: Text('Bouton 2'), // ElevatedButton
25                SizedBox(height: 16), // Espacement entre les boutons
26              ElevatedButton(
27                onPressed: null,
28                child: Text('Bouton 3'), // ElevatedButton, <Widget>[], Column, Center, Scaffold, MaterialApp
29              )
30            ]
31          )
32        )
33      )
34    )
35  }

```

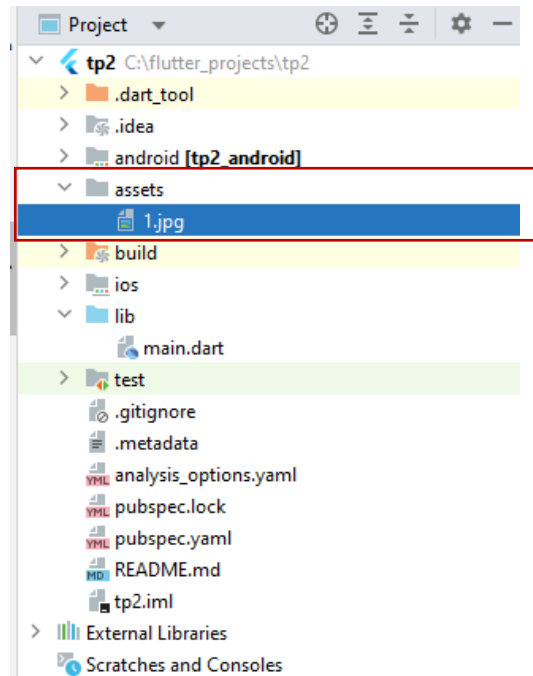
- Exécutez le code ci-dessous afin de pouvoir insérer un bouton (en état : désactivé) :

```
main.dart x widget_test.dart x
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(MyApp());
5 }
6
7 class MyApp extends StatelessWidget {
8   @override
9   Widget build(BuildContext context) {
10    return MaterialApp(
11      home: Scaffold(
12        appBar: AppBar(
13          title: Text('Exemple de centrage de bouton'),
14        ), // AppBar
15        body: Center(
16          child: ElevatedButton(
17            onPressed: null,
18            child: Text('Cliquez-ici'),),),),); // ElevatedButton, Center, Scaffold, MaterialApp
19    }
20 }
```

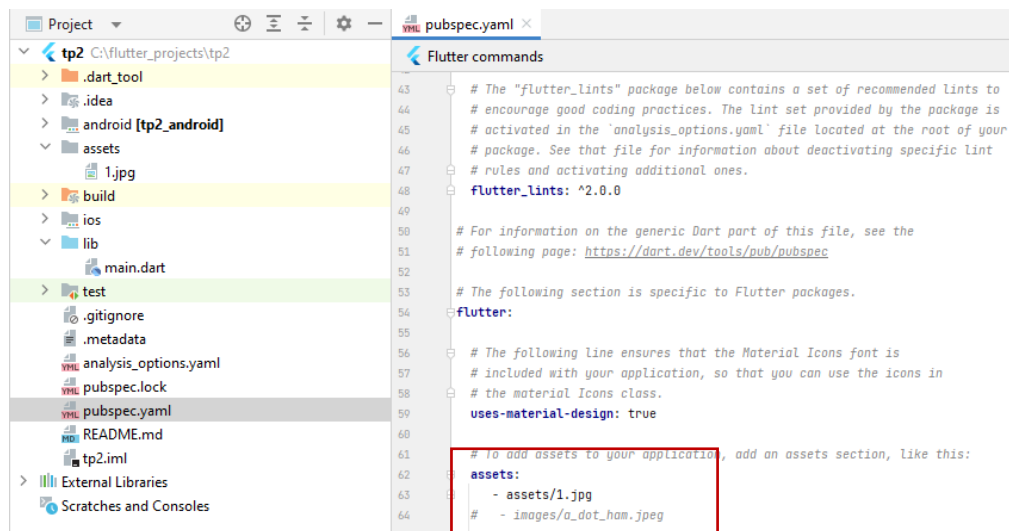
- Ajoutez une fonction paramètre « onPressed » du bouton en exécutant le code suivant :

```
main.dart x widget_test.dart x
1 import 'package:flutter/material.dart';
2
3 void main() {
4   runApp(MyApp());
5 }
6
7 class MyApp extends StatelessWidget {
8   @override
9   Widget build(BuildContext context) {
10    return MaterialApp(
11      home: Scaffold(
12        appBar: AppBar(
13          title: Text('Exemple de centrage de bouton'),
14        ), // AppBar
15        body: Center(
16          child: ElevatedButton(
17            onPressed: () {
18              print("C'est ma première Application");
19            },
20            child: Text('Cliquez-ici'),),),),); // ElevatedButton, Center, Scaffold, MaterialApp
21    }
22 }
```


- Insertion d'une image dans une application Flutter :
- Créez un dossier assets et mettez dedans une image (format .jpg) :



- Ajoutez dans la section flutter de votre fichier pubsec.yaml , le chemin relatif à votre image :



- Exécutez le code suivant :

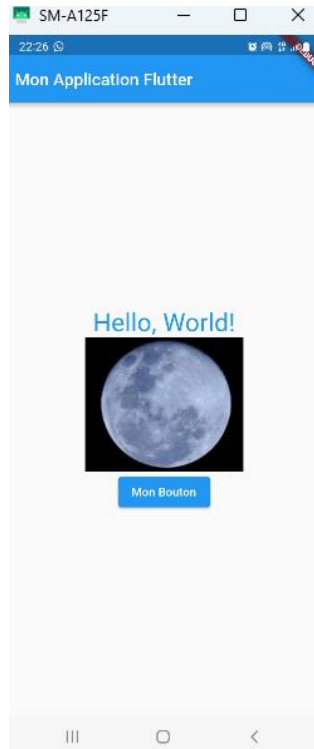
```
main.dart x
1  import 'package:flutter/material.dart';
2
3  >> void main(){
4      runApp(MyApp());
5  }
6
7  class MyApp extends StatelessWidget{
8      const MyApp({super.key});
9
10     @override
11     Widget build(BuildContext context){
12         return MaterialApp(
13             home: Scaffold(
14                 appBar: AppBar(
15                     title: Text('Image Assets'),
16                 ), // AppBar
17                 body: Center(child: Image.asset('assets/1.jpg')),
18             ) // Scaffold
19         ); // MaterialApp
20     }
21 }
```

Application à réaliser en utilisant toutes les notions vues précédemment

Créez une application qui contient :

- un widget Text personnalisé avec la couleur Rouge et une taille de 30,
- une image d'un chat,
- un bouton.

→ Voir Exemple ci-dessous :



Quizz

Question 1 - Quels éléments clés sont impliqués dans le lancement du processus de dessiner une UI dans l'écran d'un appareil ?

- a. Le dossier principal et la fonction runApp()
- b. La fonction « runApp() » et le Widget « Main »
- c. La fonction « main() » et le widget « App() »
- d. La fonction « main() » et la fonction « runApp() »

Question 2 - Quel est le rôle de la méthode « build() » ?

- a. La méthode « build() » retourne des widgets (un arbre de widget) qui devrait être dessinés sur l'écran
- b. La méthode « build() » crée une nouvelle variable
- c. La méthode « build() » retourne l'emplacement d'un widget dans l'arbre des widgets du programme.

Question 3 - Qu'est-ce qu'un widget ?

- a. C'est un nom alternatif pour une application Flutter.
- b. Les widgets sont les blocs de code construisant les interfaces utilisateur de Flutter.
- c. Les widgets sont des caractéristiques de de DART.