

@

Développement Mobile

Native Cross Platform : Flutter

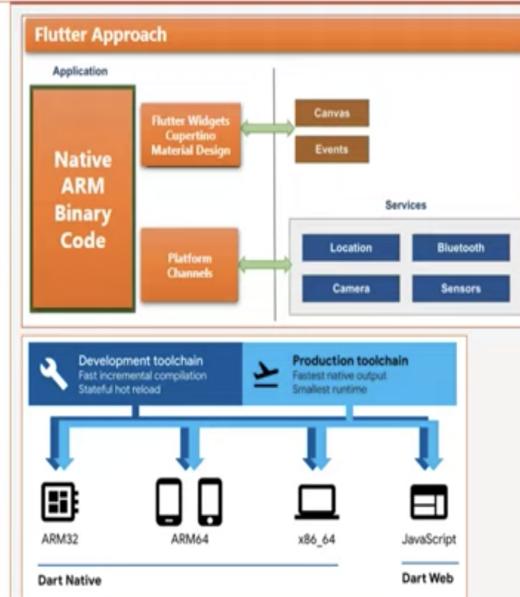
Cours I

I. Introduction

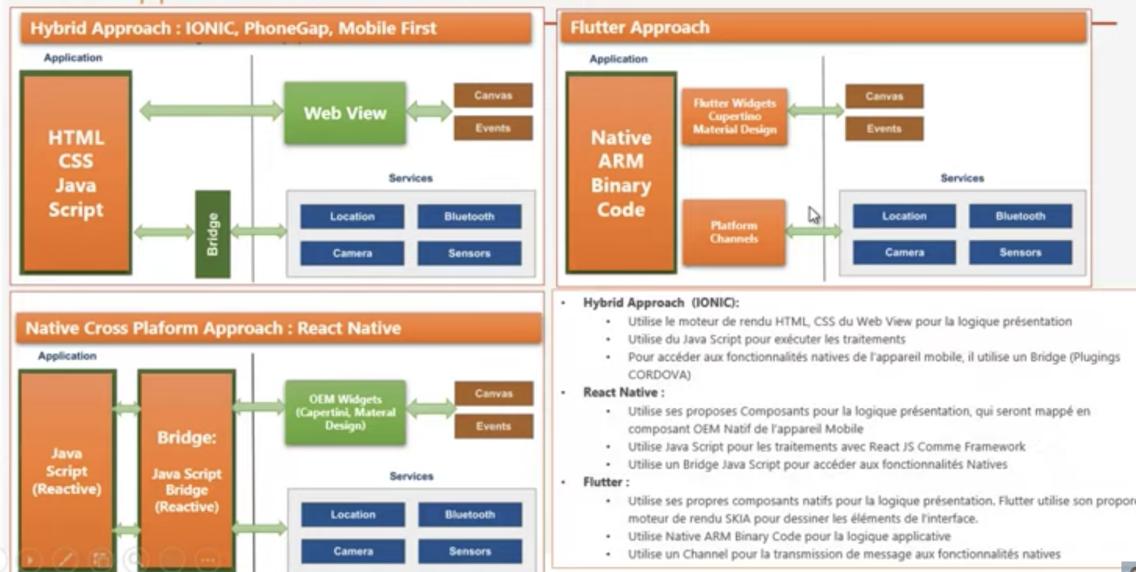
A. Qu'est ce que Flutter

Flutter ?

- Flutter est un Framework développé par Google, qui permet de créer des applications natives **Android et/ou IOS**. En utilisant le langage de programmation **DART**
- La version 1.0 de Flutter a été annoncé le 4 décembre 2018.
- **Dart** est un langage de programmation développé par la communauté Google. La première version date de 2011.
- Le but du développement de ce langage est de remplacer Javascript afin d'éviter les limites de performance de ce dernier.
- Dart a trouvé sa popularité à travers Flutter qui offre une manière typiquement différente pour développer les applications Mobile Cross Platform telle que IONIC et React Native



Développement Mobile Cross Platform



Et, le principal argument de vente de Flutter Tech sont deux choses :

- a) **Application haute performance** : les applications développées à l'aide de Flutter sont très expressives et disposent d'une interface utilisateur flexible. Son développement rapide dû au rechargeement à chaud donne vie à l'application et son expressivité offre des fonctionnalités qui sont adaptées aux expériences natives de l'utilisateur final.
- b) **Interface utilisateur expressive et flexible** : Flutter permet aux développeurs de créer facilement de belles applications à l'aide de widgets de matériaux pré-construits. Même si de nombreux widgets sont prédéfinis, Flutter permet une personnalisation complète du widget.
- c) **Développement rapide et rechargeement à chaud** : le rechargeement à chaud fait référence à l'injection de nouvelles versions des fichiers que vous avez modifiés au moment de l'exécution tout en maintenant l'application en cours d'exécution.

Les avantages et les inconvénients de Flutter :

1. **Avantages:**

- a. Flutter utilise une seule base de code, appelée Dart pour les deux plates-formes, Android et iOS, qui est un langage simple assurant la sécurité des types.
- b. Le langage Flutter et la communauté se développent à grande vitesse, publiant de nouvelles fonctionnalités, widgets et modules complémentaires.
- c. Flutter a son propre ensemble de widgets plutôt que d'utiliser les widgets fournis par le système d'exploitation hôte, ce qui signifie que l'utilisateur fournit son propre modèle de reconnaissance gestuelle, ayant ainsi un meilleur contrôle sur le rendu précis ou la personnalisation des widgets.
- d. Le rechargeement à chaud change la donne dans la productivité du processus de développement. Cela donne un effet vivant à l'application en cours de développement, rendant ainsi l'ensemble du cycle de développement plus excitant pour le développeur UI/UX utilisant Flutter.
- e. Flutter n'est pas lié à la ROM avec le système de widgets. Ainsi, il améliore sa portabilité sur un large éventail de versions d'Android et réduit ainsi ses dépendances à la plate-forme hôte.
- f. Dart et Flutter s'unissent étroitement pour optimiser la machine virtuelle Dart (VM) pour les mobiles dont Flutter a spécifiquement besoin.
- g. Flutter est un acteur établi dans le domaine du développement d'applications multiplateformes avec un soutien communautaire incroyable.

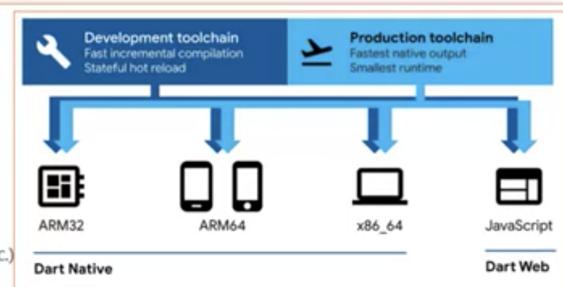
2. Les inconvénients:

En réalité, il n'y a pas d'inconvénients à flutter car il n'y a pas d'autre framework aussi efficace et élaboré que flutter. Même si nous devons en énumérer, cela serait lié au langage de programmation Dart, car lors de la conversion de Dart en JavaScript, il y a quelques bogues à corriger, Dart n'a pas de cadre pour le backend, etc.

B. Qu'est ce que dart

DART

- Dart est utilisé pour écrire
 - Des scripts simples
 - ou des applications complètes
 - Mobile (Android et IOS)
 - Web,
 - Script de ligne de commande
 - Application côté serveur,
- **Dart Native:** 
 - Pour les applications ciblant les appareils (mobile, DeskTop, Serveur, etc.)
 - Comprend à la fois une machine virtuelle Dart avec compilation
 - **JIT** (Just In Time)
 - **AOT** (Ahead-of-time)) pour produire du code machine.
- **Dart Web:**
 - Pour les applications Web, Dart Web comprend à la fois
 - Un compilateur **dartdevc** (Développement Time Compiler) : Permet d'exécuter et de débugger les applications web dart sur le moteur Chrome en utilisant WebDev Serve tool.
 - un compilateur **dart2js** (Production Time Compiler)
 - Pour faire du Dart Web :
 - **Flutter Framework**
 - **AngularDart**



II. Les Bases de Dart

A. DartPad

B. Les Types de Base

L'une des caractéristiques les plus fondamentales d'un langage de programmation est l'ensemble des types de données qu'il prend en charge. Ce sont les types de valeurs qui peuvent être représentées et manipulées dans un langage de programmation.

Le langage Dart prend en charge les types suivants:

- **Numbers**
- **Strings**
- **Booleans**

- d) **Nombres** : sont des entiers ou réels
- **entier => int**
 - **réels => double, float**
- e) **Strings:** Les chaînes représentent une séquence de caractères. Le mot-clé **String** est utilisé pour représenter des chaînes littérales. Les valeurs de chaîne sont incorporées entre guillemets simples(' . . . ') ou doubles(" . . . ").
- f) **Booleans** : Le type de données Boolean représente les valeurs booléennes **true et false**. Dart utilise le mot clé **bool** pour représenter une valeur booléenne.
- g) **void** : Un type spécial qui indique une valeur qui n'est jamais utilisée. Les fonctions qui ne renvoient pas explicitement de valeur ont le **void** type de retour.

C. Constantes et Variables

1. Constantes

a) Définition

- **const CSTE=valeur;**
- **final type CSTE=valeur;**

b) Exemples

- **const PI=3.14; print(PI);**
- **final String SRC_IMAGES="assets/images"; print(SRC_IMAGES);**

c) Exemples

2. Variables

a) Définition

type variable[=valeur]

b) Exemples

- **Entier** => int age=12; print(age);
- **Réel** => double taille=13.5; print(taille);
- **Boolean** => bool echec=true; print(echec);
- **String** => String message="Bonjour"; print(message);
- **Quelques Fonctions des chaînes**
 - String message='Bonjour';
 - print("Taille \${ message.length}");
 - print("Minuscule \${message.toLowerCase()}");
 - print("Majuscule \${ message.toUpperCase()}")

NB :

- **print()** : est une fonction pratique qui affiche la sortie.
- **Un littéral de chaîne : \$variableName ou \${expression}**
- **main()** : La fonction spéciale, *requete*, de niveau supérieur où l'exécution de l'application démarre.

D. Les Collections

Les listes et les Map sont des types de données utilisées pour représenter une collection d'objets.

La bibliothèque **dart: core** permet la création et la manipulation de ces collections via les classes List et Map prédéfinies respectivement.

1. List

a) Définition

Une liste est un groupe ordonné d'objets. Le type de données List dans Dart est synonyme du concept de tableau dans d'autres langages de programmation.

b) Exemples

```

//Liste
List <String> lString =new List();
lString.add("Bonjour");
lString.add("à");
lString.add("Tous ");
lString.add(", Au Revoir ");

print("Taille ${lString.length}"); Bookmarks
chrome://bookmarks

print("Premier Element ${lString.elementAt(0)}");

print("Boucle for , Affichage de la Liste");

for(int i=0; i< lString.length;i++){
    print(lString.elementAt(i));
}
lString.removeAt(0);

lString[2]="Je me nomme Birane Baila Wane ";

print("Methode forEach , Affichage de la Liste");

lString.forEach((element){
    print(element);
});
```

Remarque :

- Initialisation de la taille de la liste : `List<int> lst = new List(3)`
- Autre Déclaration `List<int> lst=[1,3,5,8]`
- Dart 2.3 a introduit **Spread opérateur** (`...`) et l'opérateur de **Spread opérateur null** (`...?`), qui fournissent un moyen concis d'insérer plusieurs valeurs dans une collection.
 - **Exemple 1** : utiliser le **Spread opérateur** (`...`) pour insérer toutes les valeurs d'une liste dans une autre liste :
 - `var liste = [1 , 2 , 3] ;`
 - `var liste2 = [0 , ... liste];`
 - **Exemple 1** : utiliser le **Spread opérateur null** (`...?`) pour insérer toutes les valeurs d'une liste(vide ou pas) dans une autre liste :
 - `var list;`
 - `var list2 = [0, ...?list];`

2. Maps

a) Définition

Le type de données Map représente un ensemble de valeurs sous forme de paires clé-valeur.

b) Exemples

```
Map<String, int> lMap=Map();
lMap[ "val1"] =12;
lMap[ "val2"] =14;
lMap[ "val3"] =15;
print(lMap);

print("Proprietes");

print("keys ${lMap.keys}");
print("Valeurs ${lMap.values}");

print("Parcours");

for(String key  in lMap.keys){
    print("${key} => ${lMap[key]}");
}

for(int value  in lMap.values){
    print("${value}");
}

print("Fonctions");

lMap.forEach((value,key){
    print("${key} => ${value}");
});
```

Remarque :

- Les Map supportent :
 - Generique
 - Spread opérateur (...) et Spread opérateur null(...?),

3. Les Dynamic

a) Définition

Dart est un langage typé facultativement. Si le type d'une variable n'est pas explicitement spécifié, le type de la variable est dynamique .

- Le mot-clé **dynamic** peut également être utilisé explicitement comme annotation de type.
- Le mot-clé **var** est un moyen pour déclarer une variable sans spécifier son type. Le type de cette variable est déterminé par sa valeur initiale .

b) Exemples

- **List**

- `var list = ["Bonjour",1,2.5,true] ; print(list);`
- `dynamic list = ["Bonjour",1,2.5,true] ;print(list);`

NB: Les listes créées sont de type Object ,`List<Object> list`

- **Map**

- `var details = {'username':'douve','password':'pass@123'}; print(details);`
- `dynamic details = {'username':'douve','password':'pass@123'};`

E.Le Type Function

1. Définition

Les fonctions sont les éléments constitutifs d'un code lisible, maintenable et réutilisable.

Dart est un véritable langage orienté objet, donc même les **fonctions sont des objets** et ont un type, **Function**. Cela signifie que les **fonctions peuvent être affectées à des variables ou transmises en tant qu'arguments à d'autres fonctions**.

Exemple :

```
void main() {
    affiche(calcul(2,5,somme));
    affiche(calcul(2,5,produit));

}

int somme(int a, int b){
    return a + b;
}
int produit(int a, int b){
    return a * b;
}
int calcul(int a,int b,Function operation){
    return operation(a,b);
}
void affiche(int result){
    print("Le resultat est ${result}\n");
}
```

2. Paramètre

Une fonction peut avoir n'importe quel nombre de paramètres .Ces paramètres peuvent être obligatoires ou facultatifs .

Les paramètres facultatifs peuvent être :

- soit par des **paramètres nommés**,
- soit par des **paramètres positionnels**
- soit par des **paramètres par défaut**

NB : Cependant on ne peut pas utiliser des paramètres nommés et positionnels dans la même fonction.

a) Paramètre Facultatif nommée

o Définition

```
void parametreFacultatifNominee(int x, {var y, dynamic z}) {  
    print("${x}");  
    print(" Paramètre Facultatif de Position ${y}, ${z}");  
}  
  
o Appel  
■ parametreFacultatifNominee(3);  
■ parametreFacultatifNominee(3,y:"Bonjour");  
■ parametreFacultatifNominee(  
    3,  
    y:[1,3,5,"Au revoir"],  
    z>{"login":"douve","pwd":"passer"}  
);
```

b) Paramètres facultatifs Position

o Syntaxe

```
void parametreFacultatifPosition(int x, [var y, dynamic z]) {  
    print("${x}");  
    print(" Paramètre Facultatif de Position ${y}, ${z}");  
}
```

o Appel

- parametreFacultatifPosition(3);
- parametreFacultatifPosition(3,"Bonjour");
- parametreFacultatifPosition()

```
3,  
[1,3,5,"Au revoir"],  
{"login":"douve","pwd":"passer"}  
);
```

- **Paramètre Facultatif avec Valeur Par Défaut**

```
void parametreFacultatifParDefaut(int x, int y=0,int z=1) {  
    print("${x}");  
    print(" Paramètre Facultatif par défaut ${y}, ${z}");  
}
```

Remarque:

- Au fur et à mesure que vous découvrez le langage Dart, gardez ces faits et concepts à l'esprit :
 - Toute variable est un objet , et chaque objet est une instance d'une classe. Les **nombres** , **les fonctions**, **les collections** et **null** sont des objets. À l'exception de null(si vous activez sound null safety), tous les objets héritent de la classe **Object**.
 -
- Dart prend en charge les types génériques, par **List<int>**(une liste d'entiers) ou **List<Object>**(une liste d'objets de n'importe quel type).

F. Function Anonyme et Arrow function

```
void main() {
    affiche(calcul(2,5,somme));
    affiche(calcul(2,5,(int a, int b){
        var prod=a * b;
        return prod;
    }));
}

int somme(int a, int b)=> a + b;

int calcul(int a,int b,Function operation){
    return operation(a,b);
}
void affiche(int result){
    print("Le resultat est ${result}\n");
}
```

G. Les Opérateurs

1. Opérateurs Arithmétiques

Operator	Meaning
+	Add
-	Subtract
- <i>expr</i>	Unary minus, also known as negation (reverse the sign of the expression)
*	Multiply
/	Divide
~/	Divide, returning an integer result
%	Get the remainder of an integer division (modulo)

2. Opérateurs d'incrémentation ou de décrémentation

Operator	Meaning
<code>++var</code>	<code>var = var + 1</code> (expression value is <code>var + 1</code>)
<code>var++</code>	<code>var = var + 1</code> (expression value is <code>var</code>)
<code>--var</code>	<code>var = var - 1</code> (expression value is <code>var - 1</code>)
<code>var--</code>	<code>var = var - 1</code> (expression value is <code>var</code>)

3. Opérateurs de comparaison

Operator	Meaning
<code>==</code>	Equal; see discussion below
<code>!=</code>	Not equal
<code>></code>	Greater than
<code><</code>	Less than
<code>>=</code>	Greater than or equal to
<code><=</code>	Less than or equal to

4. Opérateurs Conversion et de test de type

Opérateur	Signification
<code>as</code>	Typecast (également utilisé pour spécifier les préfixes de bibliothèque)
<code>is</code>	Vrai si l'objet a le type spécifié
<code>is!</code>	Vrai si l'objet n'a pas le type spécifié

Exercice Application :

H. La POO en Dart

Dart est un langage orienté objet avec des classes et un héritage basé sur les mixin. Chaque objet est une instance d'une classe, et toutes les classes sauf **Null** descendent de **Object**.

L'héritage basé sur **Mixin** signifie que bien que chaque classe (à l'exception de la **classe supérieure**, **Object?**) ait exactement une superclasse, un corps de classe peut être réutilisé dans plusieurs hiérarchies de classes.

Les méthodes d'extension sont un moyen d'ajouter des fonctionnalités à une classe sans modifier la classe ni créer de sous-classe.

Règles de Définition des Classes

En Dart :

- Les attributs et méthodes d'une classe ont par défaut la visibilité publique.
- Contrairement à Java, Dart n'a pas les mots-clés **public**, **protected** et **private**. Si un identifiant commence par un trait de soulignement (_), il est privé à sa bibliothèque.
- Les attributs précédés d'un **underscore(_)** sont à **private**, pour y accéder on définit des getters et des setters. **Les Getters et Setters sont généralement définis sous forme de arrow fonction.**
- Si vous activez la sécurité **null**, les variables ne peuvent pas contenir **null** à moins que vous ne disiez qu'elles le peuvent. Vous pouvez rendre une variable nullable en mettant un point d'interrogation (?) à la fin de son type. Par Exemple
 - Les attributs d'une classe doivent être initialisés ou bien mis à la sécurité nulle, **int age =0;** ou **int? age;**
- Les variables d'instance peuvent être **final** mais pas **const**.
- Les variables et méthodes **static** sont précédées du mot clé **static**.
- Pour redéfinir une méthode, on utilise annotation **@override**

1. Définition d'une Classe

a) Attributs

```
class Person {  
  
    String? x; // Déclaration d'une variable public d'instance name, initialise à null.  
    int? age; // Déclaration d'une variable private d'instance age, initialise à null.  
    late int anneeNaiss =4;/// Déclaration d'une constante public avec initialisation différée  
    final int nbreMaxEnfant=4;// Déclaration d'une constante(instanciation) public nbreMaxEnfant  
    static int nbrePersonne=0;//Déclaration d'une variable public de classe nbrePersonne  
                           //initialise à 0  
}
```

b) Le Constructeur par défaut

```
Person(){  
}  
}
```

c) Le Constructeur avec argument

```
Person(String nom,int age ){  
    this._age=age;  
    this.nom=nom  
}
```

d) Le Constructeur avec named parameter

```
Person({required this.nom}){ }
```

NB: Les attributs publics peuvent être utilisés comme paramètres nommées dans le constructeur.

Le mot clé **required** indique l'attribut est obligatoire

e) Le Factory Constructor

```
|
```

```
Person(this.anneeNaiss,this.nbreMaxEnfant){ }
```

NB: Les constantes publiques peuvent être utilisées comme paramètres dans le constructeur, on parle de factory constructeur.

Remarque :

Certaines API, notamment les constructeurs de **widgets Flutter**, n'utilisent que des **paramètres nommés, même pour les paramètres obligatoires** en utilisant le mot clé **required**.

f) Getters and Setters

```
//Getters and Setters
int? get Age => this._age;
void set Age(int? age){
    this._age=age;
}
```

g) Résumé

```
class Person {

    String? nom; // Declaration d'une variable public d'instance name, initialise a null.
    int? _age; // Declaration d'une variable private d'instance age, initialise a null.
    late int anneeNaiss =4;/// Declaration d'une constante public avec initialisation differee
    final int nbreMaxEnfant=4;// Declaration d'une constante(instance) public nbreMaxEnfant
    static int nbrePersonne=0;//Declaration d'une variable public de classe nbrePersonne
                           //initialise a 0
    Person({required this.nom}){_
        //Getters and Setters
        int? get Age => this._age;
        void set Age(int? age){
            this._age=age;
        }

        @override
        toString()=> "Nom : $nom Age:$_age";

        //Methode Static
        Person.affiche() {
            print('Je suis une Personne ');
        }
    }
}
```

2. Manipulation des Objets

```
void main() {
    Person pers=new Person(nom:"Birane Baila Wane");
    pers.Age=12;
    print(pers);
    Person.affiche();
}
```

3. Relation entre Classe

a) Heritage

```
class Employee extends Person {  
  
    Employee({nom}):super(nom:nom);  
    Employee.affiche() : super.affiche() {  
        print('Je suis un Employe');  
    }  
}
```

```
//Employee  
  
Employee emp=new Employee(nom:"Birane Baila Wane");  
Employee.affiche();
```

b) ManyToOne

```
class Fonction{  
    String? libelle;  
    Fonction({this.libelle});  
    @override  
    toString()=> "Fonction :$libelle";  
}  
class Employee extends Person {  
  
    Fonction? fonction;  
    Employee({nom,this.fonction}):super(nom:nom);  
    Employee.affiche() : super.affiche() {  
        print('Je suis un Employe');  
    }  
  
    @override  
    toString()=> super.toString()+" $fonction";  
}
```

```
Employee emp=Employee(  
                        nom:"Birane Baila Wane",  
                        fonction:Fonction(  
                            libelle:"Service Informatique"  
                        ));  
emp.Age=35;  
print(emp);  
Employee.affiche();
```

```
Application
void main() {
    List<int> numbers=[1,2,5,12,];
    List valeurs=[1, 1.3,"Bonjour";

    //Parcours de la liste
    /*
        *for(int i =0;i<valeurs.length;i++){
            print("Valeur ${valeurs[i]}");
        }

        for(var nbre in numbers){
            print("Valeur ${nbre}");
        }

    numbers.forEach((valeur){
        print("Valeur $valeur");
    });

    */
    Map<String,int> IMap={
        "cl1":1,
        "cl2":2,
    };
    /*
    IMap.forEach((key,valeur){
        print("Valeur $valeur et la cle est $key");
    });
    IMap.forEach((key,valeur)=> print("Valeur $valeur et la cle est
    $key"));
    */
    Map<String,int> IMap1=Map();
    IMap1["cl1"]=1;
    IMap1["cl2"]=2;

    var test={
```

```

    "cl1":1,
    "cl2":2,
};

var test1=[

  1,"bonjour","Au revoir",true,{
    "cl1":1,
    "cl2":2,
  }
];
var listeMap=[

  {
    "cl1":1,
    "cl2":2,
  },
  {
    "cl1":1,
    "cl2":2,
  }
];
/*
listeMap.forEach((element){

  print("Valeur de Cle1: ${element['cl1']} et Valeur de
Cle2:${element['cl2']}");

});
*/
//Fonctions Nommees avec argument obligatoire
void somme(int a ,int b){

  int s=a+b;
  print("La somme est $s");


}

int produit(var a ,int b){

  return a*b;
}

//Appeles
//somme(2,5);

```

```
//int p=produit(15.5,12);

//Fonctions Nommees avec argument faultatifs nommes
//type? : cela signifie qu'il est nullable
//int? c; c=12, c=null
//Arguments nommees {type? arg, type? arg1,}
void somme1(int a ,int b,{int? z,required int d}){
    int s;
    if(z==null) s=a+b; else s=a+b+z;
    print("La somme est \$s");

}

//Appel de la fonction

somme1(1,2,d:15);
somme1(1,2,d:20,z:16);
```

//POO Dart

```
Fonction f=Fonction(libelle:"DW");
Person p1=Person(nom:"Amadou Diop",fonction:f);
```

```
Fonction f1=Fonction(libelle:"RX");
Person p2=Person(nom:"Amadou Diop",age:12,fonction:f1);
```

```
Person p2=Person(
    nom:"Amadou Diop",
    age:12,
    fonction:Fonction(
        libelle:"DW",
        nbreEmp:23,
        service:Service(
            nom:"Info",
```

```

        produits:[
            Produit(id:1,libelle:"produit1"),
            Produit(id:2,libelle:"produit2",qteStock:12)
        ]
    ))
);

}

/*
 * Tous les attributs d'une classe doivent être initialisés ou misent à nullables
 * Attributs sont par défaut à public
 * Attributs privés _nomAttribut
 *
 */
class Person{
    //Attributs Instances
    String nom="";
    int? age;
    final int taxe=18;

    //Attributs Statics
    static int nbrePersonne=1;

    //ManyToOne
    Fonction fonction;

    //Constructeur Par défaut
    /*Person({required String nom,int? age}){
        this.nom=nom;
        this._age=age;
    }*/
    Person({required this.nom,this.age,required this.fonction}){}

    //Getters et Setters
    /*int? get Age=> this._age;
    void set Age(int? age){*/

```

```

    this._age=age;
}*/
void set Age(int? age)=>this._age=age;/*
}

class Fonction {
    String libelle;
    int? nbreEmp;
    Service? service;
    Fonction({required this.libelle,this.nbreEmp,this.service}{})
}

class Service {
    String? nom;
    List<Produit>? produits=[];
    Service({required this.nom,this.produits}{})
}

class Produit{
    int id;
    String libelle;
    double? qteStock;
    Produit({required this.id,this.qteStock,required this.libelle}{})
}

class Employe extends Person{

    String adresse;
    Employe({nom,fonction,age,required
    this.adresse}):super(nom:nom,age:age,fonction:fonction){

}

```

}

III. Flutter

Flutter est le **SDK mobile de Google** permettant de créer des applications iOS et Android natives à partir d'une seule base de code.

Lors de la création d'applications avec Flutter, tout se tourne vers les Widgets ou les blocs avec lesquels les applications Flutter sont construites.

L'interface utilisateur de l'application est composée de nombreux widgets simples, chacun d'eux gérant une tâche particulière. C'est la raison pour laquelle les développeurs Flutter ont tendance à considérer leur application Flutter comme un **arbre de widgets**.

Par rapport à ses technologies contemporaines telles que **React Native, Kotlin et Java**, Flutter est bien meilleur en ce qui concerne la base de code unique pour Android et iOS, une interface utilisateur réutilisable et une logique métier, une compatibilité, des performances et une productivité élevées.

A. Installation de l'environnement

1. Prérequis sur windows

- Un ordinateur supportant la virtualisation (pour bénéficier des émulateurs)
- Windows 10, 8 ou 7.
- Une connexion internet (pour télécharger les outils nécessaires)

2. Téléchargement du SDK de Flutter

Accéder au site flutter.dev/ « Get started » /« Install » s'affiche.

Cliquez sur Windows. On arrive sur la documentation que l'on va beaucoup utiliser :

The screenshot shows the Flutter website's navigation bar at the top with links for Docs, Showcase, Community, and a search bar. Below the navigation is a banner with the text "Curious about how Flutter is designed? See [Flutter's architectural overview](#)". The main content area has a sidebar on the left with sections like "Get started", "From another platform?", "Samples & tutorials", and "Development". The main content area title is "Windows install" with a subtitle "System requirements". It includes a list of requirements and a note about Git for Windows.

This screenshot shows the "Get the Flutter SDK" page. The sidebar on the left is identical to the previous one. The main content area title is "Get the Flutter SDK". It contains instructions for downloading the installation bundle, extracting it, and running the Flutter command. A warning box states "⚠ Warning: Do not install Flutter in a directory like C:\Program Files\ that requires elevated privileges." Below this, there's a code snippet for cloning the Flutter repo from GitHub. The right sidebar contains a "Contents" section with links to various setup steps.

This screenshot shows the "Update your path" guide. The sidebar on the left is identical. The main content area title is "Update your path". It explains how to add Flutter to the PATH environment variable. A note mentions that the Dart command is included in the Flutter SDK. A code snippet shows the "which flutter dart" command. The right sidebar contains a "Contents" section with links to various setup steps.

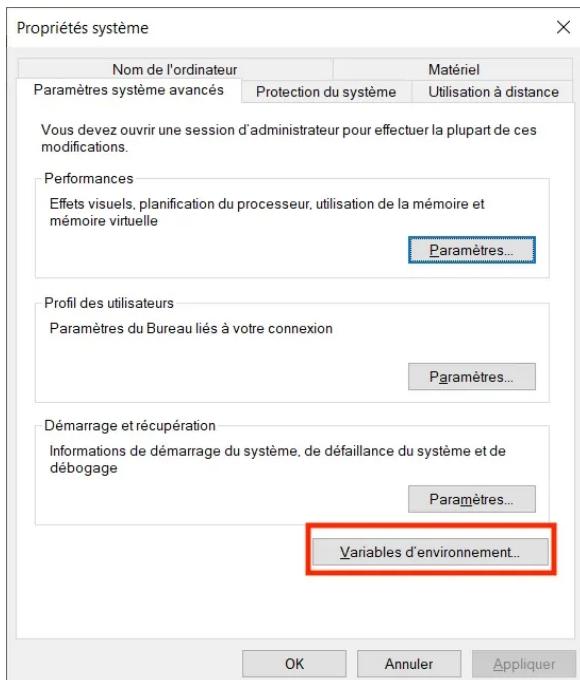
3. Ajout des variables d'environnement

This screenshot provides more detail on updating the PATH. It includes a note about the Dart command being included in the Flutter SDK. A code snippet shows the "which flutter dart" command. Another note explains that the two commands don't come from the same bin directory and provides a command to update the path. The right sidebar contains a "Contents" section with links to various setup steps.

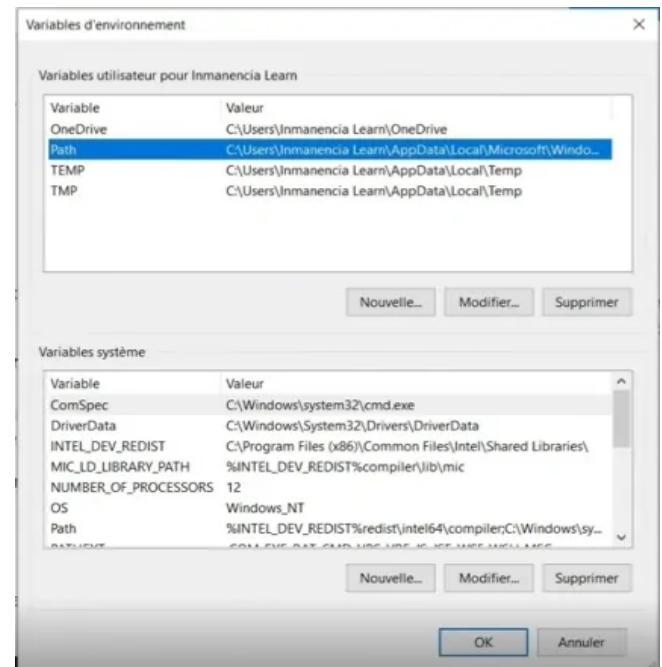
a. Tester l'existence de la commande flutter

```
C:\Users\Inmanencia Learn>flutter  
'flutter' n'est pas reconnu en tant que commande interne  
ou externe, un programme exécutable ou un fichier de commandes.  
  
C:\Users\Inmanencia Learn>
```

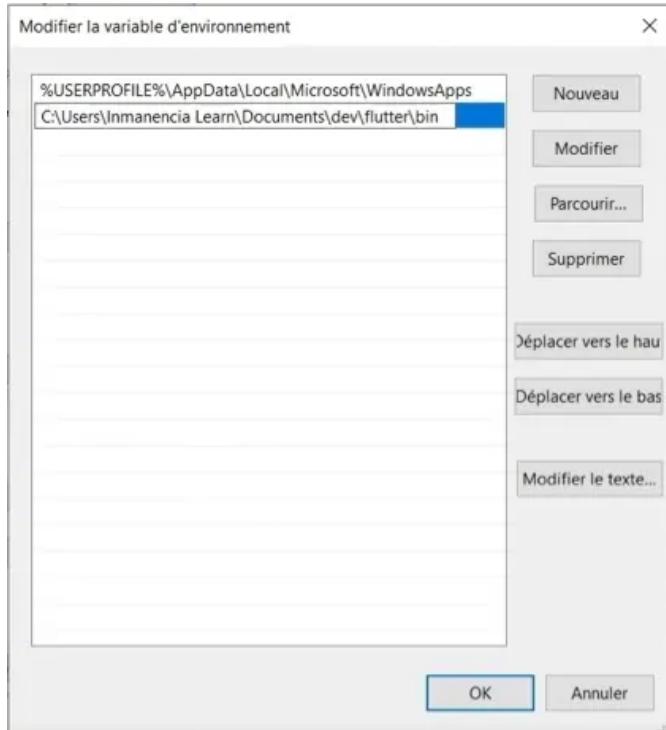
b. Ajout du SDK dans le Path



Etape 1



Etape 2



Etape 3

c. Test de la commande flutter

```
C:\ Invité de commandes - flutter
Microsoft Windows [version 10.0.17763.678]
(c) 2018 Microsoft Corporation. Tous droits réservés.

C:\Users\Inmanencia Learn>flutter
Checking Dart SDK version...
Downloading Dart SDK from Flutter engine fee001c93f25a1e7258e762781a7361f122d29f5...
```

4. Installation du SDK via Android Studio

a) Vérification de l'installation du SDK android

Tapez la commande : **flutter doctor**

```
Downloading package sky_engine...
Downloading common tools...
Downloading common tools...
Downloading windows-x64 tools...
Doctor summary (to see all details, run flutter doctor -v):
[!] Flutter (Channel stable, v1.7.8+hotfix.4, on Microsoft Windows [version 10.0.17763.6781], locale fr-FR)
  X Android toolchain - develop for Android devices
    X 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/setup/#android-setup for detailed instructions).
      If the Android SDK has been installed to a custom location, set ANDROID_HOME to that location.
      You may also want to add it to your PATH environment variable.

[!] Android Studio (not installed)
[!] Connected device
  ! No devices available

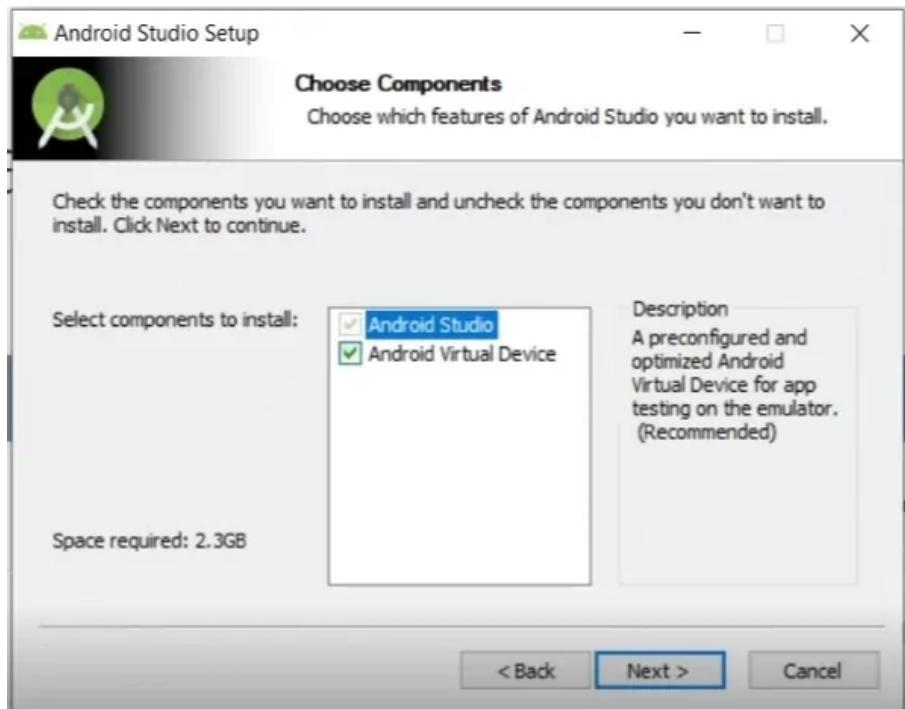
I Doctor found issues in 3 categories.

C:\Users\Inmanencia Learn>
```

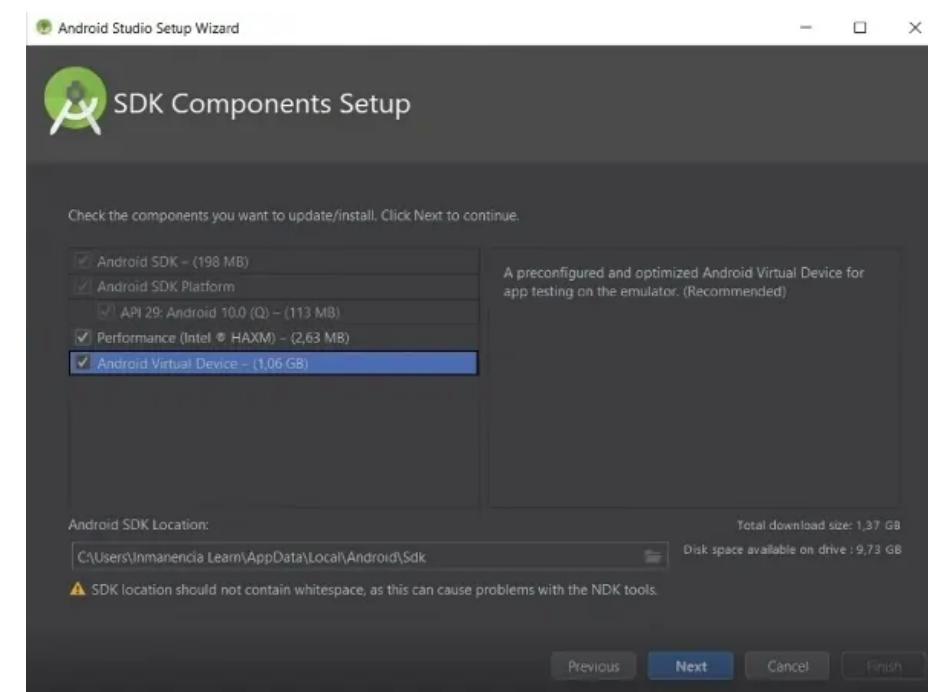
Ce résultat montre que le SDK via Android Studio n'est pas installé

b) Installation du SDK android

Lien de Telechargement : <https://developer.android.com/studio/index.html#downloads>



Etape 1



Etape 2

```
Invite de commandes - flutter

[✓] Flutter (Channel stable, v1.7.8+hotfix.4, on Microsoft Windows [version 10.0.17763.678], locale fr-FR)
[✗] 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/setup/#android-setup for detailed instructions).
      If the Android SDK has been installed to a custom location, set ANDROID_HOME to that location.
      You may also want to add it to your PATH environment variable.

[!] Android Studio (not installed)
[!] Connected device
    ! No devices available

! Doctor found issues in 3 categories.

C:\Users\Inmanencia Learn>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, v1.7.8+hotfix.4, on Microsoft Windows [version 10.0.17763.678], locale fr-FR)

[!] Android toolchain - develop for Android devices (Android SDK version 29.0.2)
    ✗ Android licenses not accepted. To resolve this, run: flutter doctor --android-licenses
[!] Android Studio (version 3.5)
    ✗ Flutter plugin not installed; this adds Flutter specific functionality.
    ✗ Dart plugin not installed; this adds Dart specific functionality.
[!] Connected device
    ! No devices available

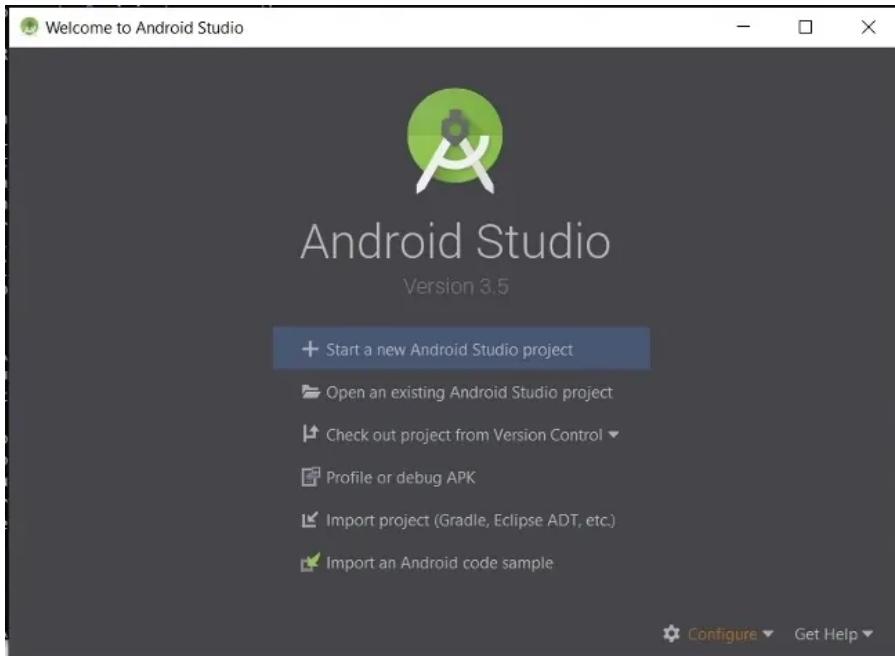
! Doctor found issues in 3 categories.

C:\Users\Inmanencia Learn>
```

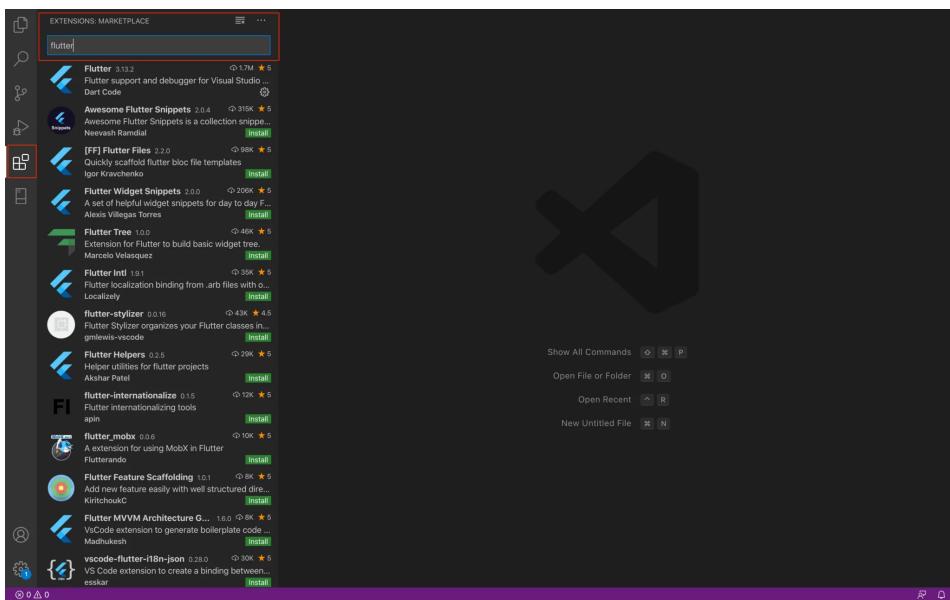
c) Acceptation des licences et ajout des plugins

Tapez la commande : **flutter doctor --android-licenses**

Il s'agit d'installer les deux plugins **Flutter** et **Dart** pour Android Studio



5. Installation sur Vscode



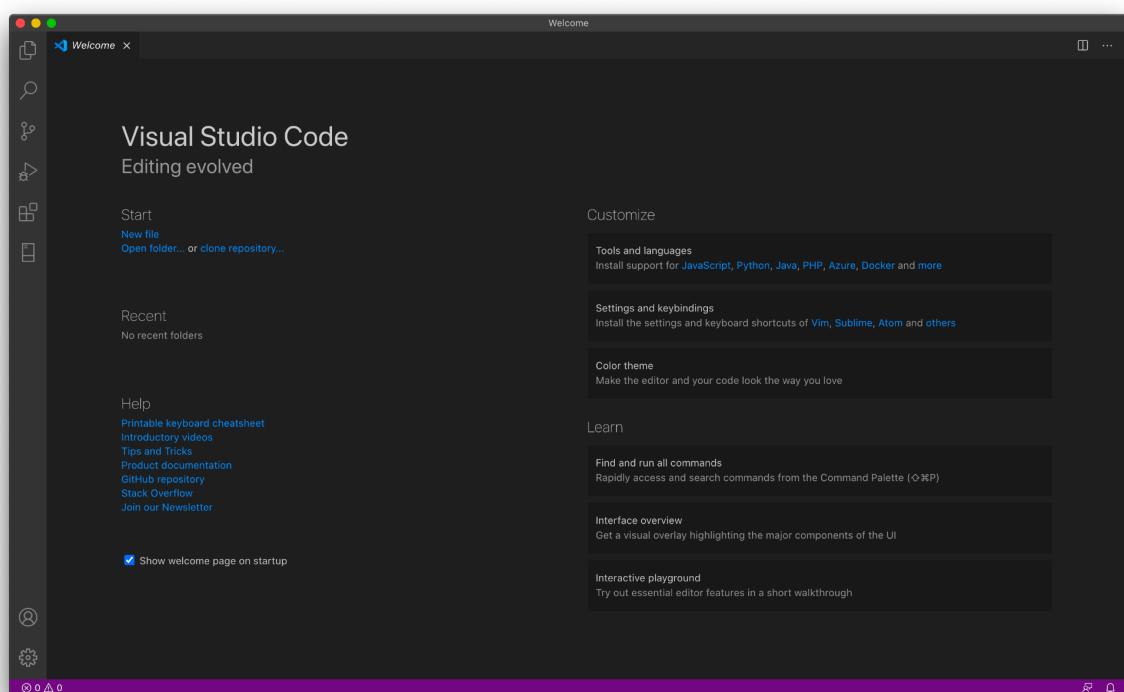
```
C:\Users\Inmanencia Learn>flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, v1.7.8+hotfix.4, on Microsoft Windows [version 10.0.17763.678], locale fr-FR)
[✓] Android toolchain - develop for Android devices (Android SDK version 29.0.2)
[✓] Android Studio (version 3.5)
[✓] VS Code (version 1.37.1)
[!] Connected device
    ! No devices available

! Doctor found issues in 1 category.

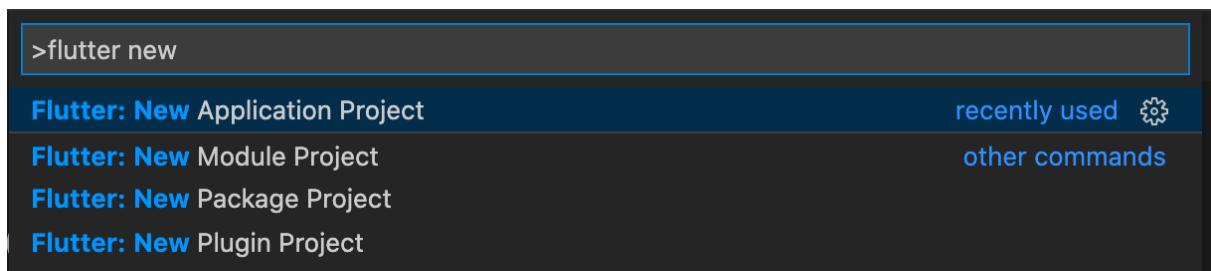
C:\Users\Inmanencia Learn>
```

B. Crédation d'un Projet Flutter

1. Etape 1

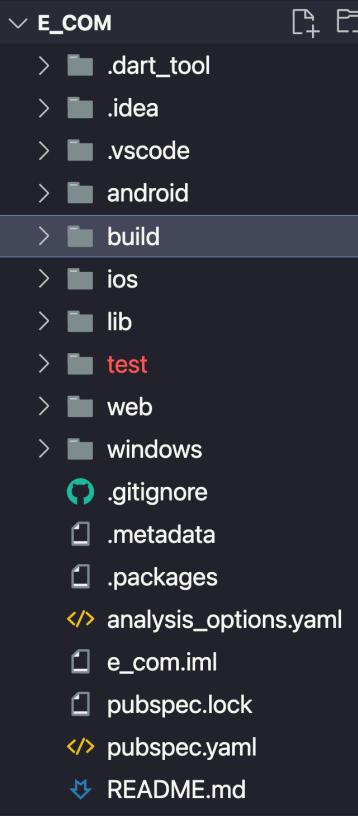


« Ctrl + Shift + P »



NB : **flutter create nomProjet**

2. Structure Projet Flutter

Structure	Description
	<ul style="list-style-type: none">● assets/images: dossier des images du projet● android: dossier de l'application android générer par flutter● ios: dossier de l'application ios générer par flutter● lib: dossier source<ul style="list-style-type: none">○ models: Entités gérées par l'application○ pages: Pages de notre application mobile○ services: Toutes les méthodes d'interaction avec la base de données○ widgets: Les composants de pages● pubspec.yaml:<ul style="list-style-type: none">○ ajout d'injection de dépendances○ déclaration des images

C. Architecture d'une application Flutter

L'application d'architecture Flutter consiste principalement en :

- Widget
- Gestures
- Notion d'État
- Couches

1. Notion de Widget

Les widgets sont le composant principal de toute application flutter. Il agit comme une interface utilisateur permettant à l'utilisateur d'interagir avec l'application.

Toute application flutter est elle-même un widget composé d'une combinaison de widgets.

Dans une application standard, la racine définit la structure de l'application suivie d'un widget **MaterialApp** qui maintient essentiellement ses composants internes en place.

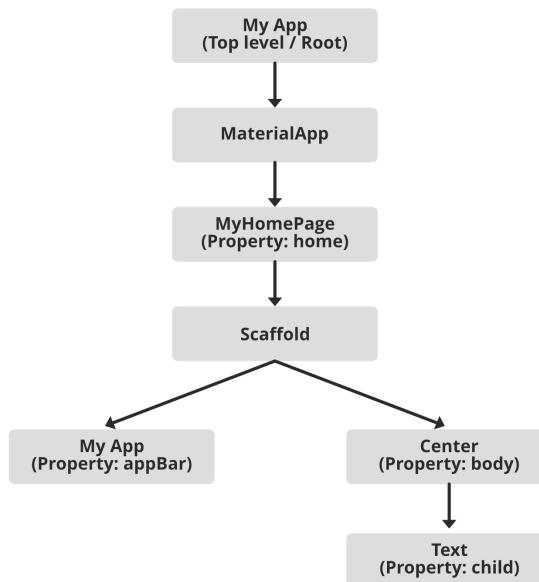
C'est là que les propriétés de l'interface utilisateur et de l'application elle-même sont définies.

Le **MaterialApp** a un widget **Scaffold** qui se compose des composants visibles (widgets) de l'application.

Le **Scaffold** a deux propriétés principales, le **body** et le **appBar** d'application.

Il contient tous les widgets enfants et c'est là que toutes ses propriétés sont définies.

Le schéma ci-dessous montre la hiérarchie d'une application flutter



2. Les Types de Widgets

On distingue deux classes de widget: **Stateless et StateFull**

a. Stateless Widget (Sans État)

```
abstract class StatelessWidget extends Widget {  
  
  const StatelessWidget({ Key key }) : super(key: key);  
  
  @override  
  StatelessElement createElement() => StatelessElement(this);  
  
  @protected  
  Widget build(BuildContext context);  
}
```

Un **stateless widget** est un widget qui ne dépend que des informations qui lui sont fournies par son parent lors de son **build**. Aucun autre événement de l'utilisateur ne relancera le build donc ses informations ne sont plus modifiables .



Exemple :

```
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',  
      debugShowCheckedModeBanner: false,  
      theme: ThemeData(  
        primarySwatch: Colors.blue,  
      ), // ThemeData  
      home: const BasicPage(),  
    ); // MaterialApp  
  }  
}
```

i. La Classe BuildContext

La classe **BuildContext** est, comme son nom l'indique, le contexte dans lequel un widget spécifique est construit.

D'une manière générale, il existe 2 cas d'utilisation du contexte :

- Interagissez avec vos parents (obtenez/publiez des données principalement)
- Une fois rendu à l'écran, obtenez la taille et la position de votre écran

Exemple :

1. Récupération de la platform

Theme.of(context).platform

2. Récupération de la Taille

MediaQuery.of(context).size

```
abstract class BuildContext {  
  
    Widget get widget;  
  
    BuildOwner get owner;  
  
    bool get debugDoingBuild;  
  
    RenderObject findRenderObject();  
  
    Size get size;  
  
    InheritedWidget dependOnInheritedElement(InheritedElement ancestor, { Object a  
    T dependOnInheritedWidgetOfExactType<T extends InheritedWidget>({ Object aspect  
    InheritedElement getElementForInheritedWidgetOfExactType<T extends InheritedWi  
    T findAncestorWidgetOfExactType<T extends Widget>();  
    T findAncestorStateOfType<T extends State>();  
    T findRootAncestorStateOfType<T extends State>();  
    T findAncestorRenderObjectOfType<T extends RenderObject>();  
    void visitAncestorElements(bool visitor(Element element));  
    void visitChildElements(ElementVisitor visitor);  
    DiagnosticsNode describeElement(String name, {DiagnosticsTreeStyle style = Diagn  
    DiagnosticsNode describeWidget(String name, {DiagnosticsTreeStyle style = Diagn  
    List<DiagnosticsNode> describeMissingAncestor({ @required Type expectedAncesto  
    DiagnosticsNode describeOwnershipChain(String name);  
}
```

ii. Les StatelessWidget Widgets de Base

1. Scaffold :

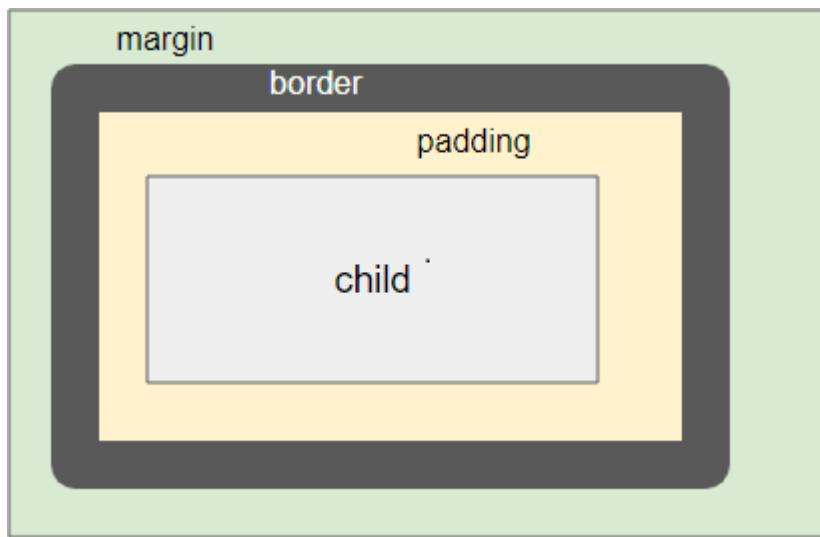
Le **Scaffold** représente le squelette de notre vue.

À l'intérieur de **Scaffold**, il y a généralement :

- **appBar**, qui, comme son nom l'indique, définit la barre d'application de l'application.
- **body**, où tous les widgets composants sont placés. C'est là que les propriétés de ces widgets sont définies. Tous ces widgets combinés forment la page de l'application elle-même. Le widget **Center** a une propriété, **Enfant**, qui fait référence au contenu réel et il est construit à l'aide du widget **Texte**.
- **drawer**

2. Container

Le **Container** est une boîte utilisée pour contenir un widget enfant. En même temps, il est possible de définir son style grâce à des propriétés telles que padding, margin, alignment, etc. Container surligne le contenu ou le sépare des autres.



Container constructor

```
Container(  
  {Key key,  
   AlignmentGeometry alignment,  
   EdgeInsetsGeometry padding,  
   Color color,  
   Decoration decoration,  
   Decoration foregroundDecoration,  
   double width,
```

```
        double height,  
        BoxConstraints constraints,  
        EdgeInsetsGeometry margin,  
        Matrix4 transform,  
        Widget child, Clip clipBehavior: Clip.none}  
)
```

3. Text

Un **Text** est un widget dans Flutter qui nous permet d' afficher une chaîne de texte avec une seule ligne dans notre application .

Exemple:

```
Text(  
  'Hello',  
  textAlign: TextAlign.center,  
  overflow: TextOverflow.ellipsis,  
  style: const TextStyle(fontWeight: FontWeight.bold),  
)
```

4. RichText et TextSpan

Un **RichText** est un widget dans Flutter qui nous permet d' afficher plusieurs textes avec des styles différents.

Un **TextSpan** est un élément de texte du **RichText**

Exemple:

```
RichText(  
  text: TextSpan(  
    text: 'Hello ',  
    style: DefaultTextStyle.of(context).style,  
    children: [  
      TextSpan(text: 'bold',  
        style: TextStyle(  
          fontWeight: FontWeight.bold  
        )),  
      TextSpan(text: ' world!'),  
    ],  
  ),  
)
```

5. Center

Un **Center** est un widget qui centre un autre widget.

6. Icon

Un **Icon** est un widget qui charge une image.

Exemple:

```
Icon(  
  Icons.favorite,
```

```
        color: Colors.pink,  
        size: 24.0,  
        semanticLabel: 'Text to announce in accessibility modes',  
    )
```

7. Les Images

Dans Flutter, nous utilisons le widget `Image` pour afficher les images.

Il prend en charge les formats d'image tels que PEG, PNG, GIF, GIF animé, WebP, WebP animé, BMP et WBMP.

La classe `Image` à des constructeurs :

1. `Image.network` - Pour afficher une image à partir d'une URL

Exemple:

```
1. Image.network(  
    'https://docs.flutter.dev/assets/images/dash/dash-fainting.gif');  
2. Container(  
    width: double.infinity,  
    height: double.infinity,  
    color: Colors.yellow,  
    child: Image.network(  
        "https://images.unsplash.com/photo-1547721064-da6cfb341d50",  
        fit: BoxFit.cover,  
    ),  
)
```

NB : On peut ajuster une image à l'intérieur d'un conteneur avec l'argument

`fit`.On peut utiliser :

- a. `BoxFit.contain` - Aussi grand que possible mais contenu dans le conteneur
- b. `BoxFit.cover` - Aussi petit que possible mais couvrant tout le conteneur
- c. `BoxFit.fill` - Remplit tout le conteneur mais peut déformer le rapport d'aspect de l'image
- d. `BoxFit.fitHeight` - La hauteur de l'image sera égale à celle du conteneur.
Cela ne gâchera pas le rapport d'aspect de l'image

- e. `BoxFit.fitWidth` - La hauteur de l'image sera égale au conteneur.
- f. `BoxFit.none` - L'image n'est pas du tout redimensionnée
- g. `BoxFit.scaleDown` - Réduit l'image pour l'adapter à l'intérieur du conteneur

2. `Image.asset` - Pour afficher l'image du groupe d'actifs

Les applications Flutter peuvent inclure à la fois du code et des actifs. Un actif est un fichier fourni et déployé avec votre application, et accessible lors de l'exécution.

Les types courants de ressources incluent les données statiques (par exemple, les fichiers JSON), les fichiers de configuration, les icônes et les images (JPEG, WebP, GIF, WebP/GIF animé, PNG, BMP et WBMP).

Les Étapes :

- a. Créer un dossier `image assets` dans le projet
- b. Déclarer les images dans le fichier `pubspec.yaml`

1. Approche 1 :

```
flutter:  
  
  assets:  
  
    - assets/my_icon.png  
  
    - assets/background.png
```

2. Approche 2

```
flutter:  
  
  assets:  
  
    - directory/  
  
    - directory/subdirectory/
```

8. Card

Un Card est un container avec des coins légèrement arrondis et une ombre en élévation.

Exemple:

1. Card avec des bords arrondis et une couleur de bordure en verte

```
Card(  
    child: ListTile(  
        title: Text("Codesinsider.com"),  
    ),  
    elevation: 8,  
    shadowColor: Colors.green,  
    margin: EdgeInsets.all(20),  
    shape: OutlineInputBorder(  
        borderRadius: BorderRadius.circular(10),  
        borderSide: BorderSide(color: Colors.green, width: 1)  
    ),  
)
```

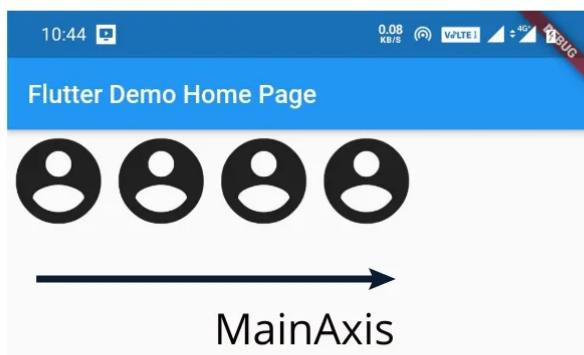
2. Card circulaire

```
Card(  
    child: Container(  
        height: 160,  
        width: 160,  
        child: Center(  
            child: ListTile(  
                title: Text("Codesinsider.com"),  
            ),  
        ),  
        elevation: 8,  
        shadowColor: Colors.green,  
        margin: EdgeInsets.all(20),  
        shape: CircleBorder(side: BorderSide(  
            width: 1,  
            color: Colors.white  
        ),  
    ),
```

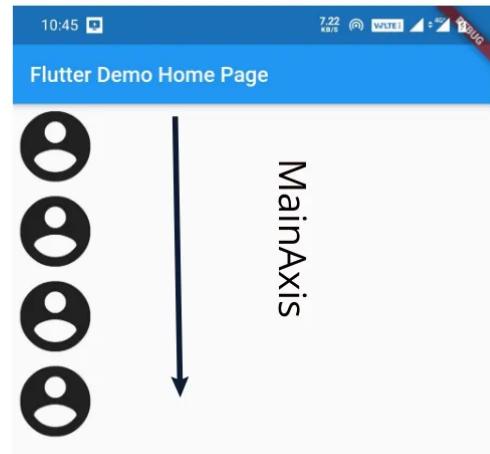
9. Column et Row

Les widgets **Column** et **Row** sont très utiles pour concevoir des mises en page Flutter . Ces layouts permettent d'ajouter plusieurs éléments.**Column** donne une orientation verticale alors que **Row** donne une orientation horizontale.

Row

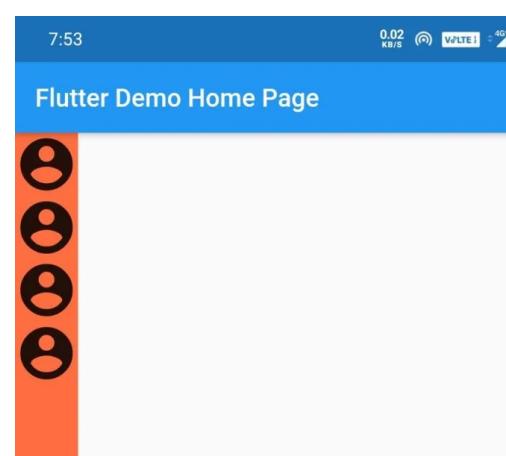


Column



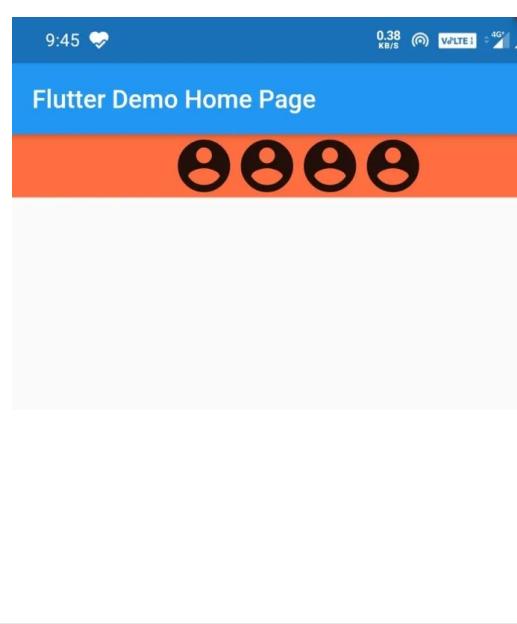
Exemple Column :

```
Container(  
  color: Colors.deepOrangeAccent,  
  child: Column(  
    mainAxisAlignment: MainAxisAlignment.start,  
    children: <Widget>[  
      Icon(Icons.account_circle, size: 50),  
      Icon(Icons.account_circle, size: 50),  
      Icon(Icons.account_circle, size: 50),  
      Icon(Icons.account_circle, size: 50),  
    ],),
```



Exemple Row :

```
Container(  
    color: Colors.deepOrangeAccent,  
    child: Row(  
        mainAxisAlignment: MainAxisAlignment.center,  
        children: <Widget>[  
            Icon(Icons.account_circle, size: 50),  
            Icon(Icons.account_circle, size: 50),  
            Icon(Icons.account_circle, size: 50),  
            Icon(Icons.account_circle, size: 50),],),),
```



10. Expanded

Le widget **Expanded** permet d'occuper l'espace restante d'un column ou d'une ligne.

11. Spacer

Le widget **Spacer** permet d'insérer de l'espace entre les éléments .

Exemple:

```
Spacer(flex: 2),
```

12. CircleAvatar et ImagePovider

Le widget **CircleAvatar** permet d'ajouter une image arrondie.

Exemple:

```
CircleAvatar(  
    radius: 110,  
    backgroundImage: NetworkImage("https://s3.o7planning.com/images/boy-128.png"),  
)
```

13. Stack

Le widget **Stack** permet d'empiler les éléments .

Exemple:

```
Stack(  
  children: <Widget>[  
    Container( width: 100, height: 100,color: Colors.red,),  
    Container( width: 80,height: 80,color: Colors.blue,),  
  ],  
)
```

14. Divider

Le widget **Divider** permet d'insérer un séparateur entre des colonnes.

Exemple:

```
Divider(height: 20,thickness: 5,indent: 20,endIndent: 0,color: Colors.black )
```

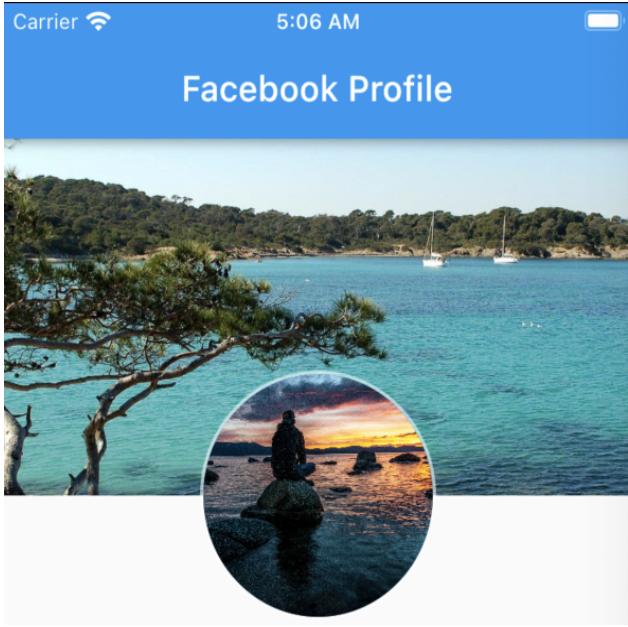
15. SingleChildScrollView

Le widget **SingleChildScrollView** qui permet de dérouler des éléments sur l'axe vertical.

Exemple:

```
SingleChildScrollView(  
  child: new Column(  
    children: [  
      ],),  
)
```

Exercice D'application: Réalisation d'une Page de Profil



```
Column textHeader() {
  return Column(children: [
    Text(
      "Birane Baila Wane",
      style: TextStyle(
        fontSize: 25,
        fontStyle: FontStyle.italic,
        fontWeight: FontWeight.bold), // TextStyle
    ), // Text
    Padding(
      padding: EdgeInsets.all(10),
      child: Text(
        "Ingenieur Informaticien:Analyste Concepteur,Developpeur FullStack",
        style: TextStyle(
          color: Colors.grey,
          fontStyle: FontStyle.italic,
        ), // TextStyle
        textAlign: TextAlign.center,
      ), // Text
    ), // Padding
  ]); // Column
}
), // Stack
); // Container
```



```
Column textHeader() {
  return Column(children: [
    Text(
      "Birane Baila Wane",
      style: TextStyle(
        fontSize: 25,
        fontStyle: FontStyle.italic,
        fontWeight: FontWeight.bold), // TextStyle
    ), // Text
    Padding(
      padding: EdgeInsets.all(10),
      child: Text(
        "Ingenieur Informaticien:Analyste Concepteur,Developpeur FullStack",
        style: TextStyle(
          color: Colors.grey,
          fontStyle: FontStyle.italic,
        ), // TextStyle
        textAlign: TextAlign.center,
      ), // Text
    ), // Padding
  ]); // Column
}
```



```
Container buttonContainer({IconData? icon, String? text}) {
  return Container(
    margin: EdgeInsets.only(left: 10, right: 10),
    padding: EdgeInsets.all(15),
    decoration: BoxDecoration(
      borderRadius: BorderRadius.circular(10), color: Colors.blue),
    child: (icon == null)
      ? Center(
        child: Text(
          text ?? "",
          style: TextStyle(color: Colors.white, fontSize: 18),
        )) // Text // Center
      : Icon(
        icon,
        color: Colors.white,
      ), // Icon
    height: 50,
  ); // Container
}
```

A propos de Mois

- Dakar, Senegal
- Developpeur Flutter
- Marie et pere de deux enfants

```
Widget sectionTitleText(String text) {
  return Padding(
    padding: EdgeInsets.all(10),
    child: Text(text,
      style: TextStyle(fontWeight: FontWeight.w600, fontSize: 18)),
  ); // Padding
}

Widget aboutRow({IconData? icon, required String text}) {
  return Row(
    children: [
      Padding(
        padding: EdgeInsets.only(left: 10),
        child: Icon(
          icon,
        ), // Icon
      ), // Padding
      Padding(padding: EdgeInsets.all(10)),
      Text(
        text,
        style: TextStyle(fontSize: 15),
      ) // Text
    ],
  ); // Row
}
```

Amis



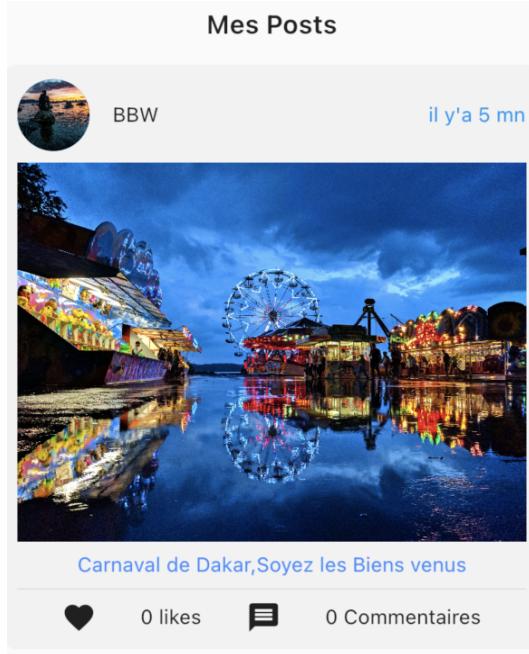
Mossa

Aissatou

Cheikhou

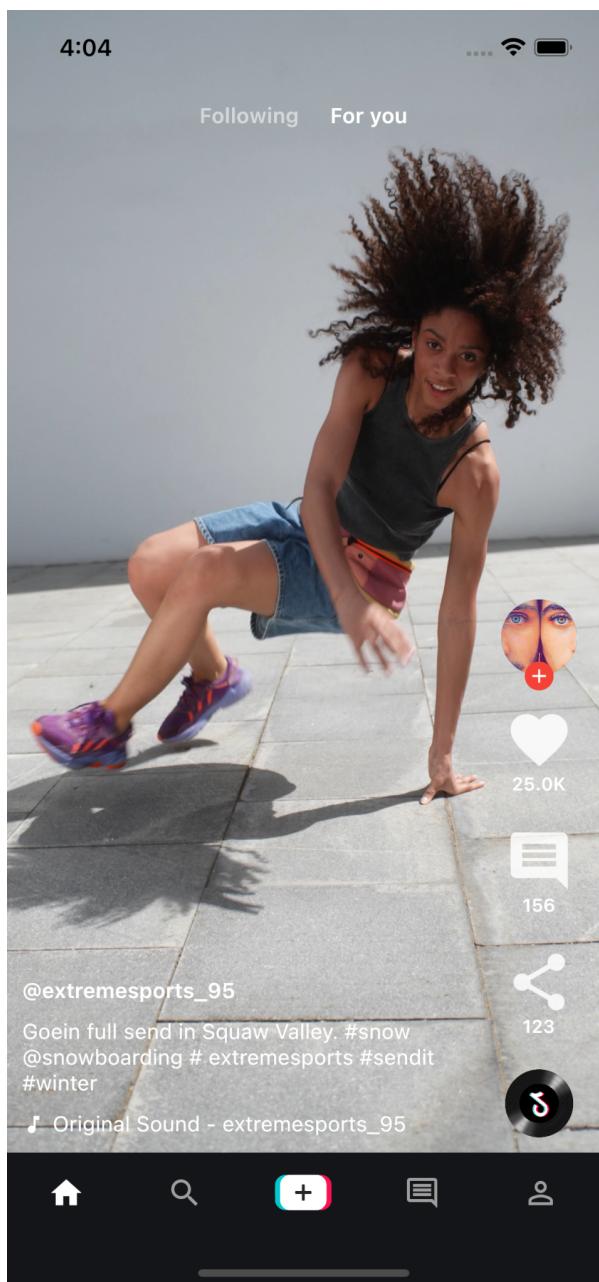
```
Column friendsImage(String name, String pathImage, double width) {
  return Column(
    children: [
      Container(
        margin: EdgeInsets.all(2),
        width: width,
        height: width,
        decoration: BoxDecoration(
          image: DecorationImage(
            image: AssetImage(pathImage), fit: BoxFit.cover), // DecorationImage
          borderRadius: BorderRadius.circular(20),
          boxShadow: [BoxShadow(color: Colors.grey)], // BoxDecoration
        ), // Container
        Padding(
          padding: const EdgeInsets.only(top: 8.0),
          child: Text(name),
        ) // Padding
      ),
    ]; // Column
}

Row allFriends(double width) {
  List<Map<String, String>> friends = [
    {"name": "Mossa", "image": "images/cat.jpg"},
    {"name": "Aissatou", "image": "images/duck.jpg"},
    {"name": "Cheikhou", "image": "images/playa.jpg"},
  ];
  List<Widget> children = [];
  for (var friend in friends) {
    String name = friend["name"] as String;
    String image = friend["image"] as String;
    children.add(friendsImage(name, image, (width - 12) / friends.length));
  }
  return Row(
    children: children,
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  );
}
```



```
Container post(
    required String time,
    required String image,
    required String desc,
    comments = 0,
    likes = 0) {
  return Container()
    margin: EdgeInsets.only(top: 8, left: 1, right: 1),
    padding: EdgeInsets.all(10),
    decoration: BoxDecoration(
      borderRadius: BorderRadius.circular(10),
      color: Color.fromRGBO(220, 220, 220, 0.25)), // BoxDecoration
  child: Column(
    children: [
      Row(
        children: [
          CircleAvatar(
            radius: 25,
            backgroundImage: AssetImage("images/profile.jpg"),
          ), // CircleAvatar
          Padding(padding: EdgeInsets.all(8.0)),
          Text("BBW"),
          Spacer(),
          Text(
            time,
            style: TextStyle(
              color: Colors.blueAccent,
            ), // TextStyle
            textAlign: TextAlign.end,
          ), // Text
          Divider(),
          Row(
            mainAxisSize: MainAxisSize.max,
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: [
              Icon(Icons.favorite),
              Text("$likes likes"),
              Icon(Icons.message),
              Text("$comments Commentaires")
            ],
          ) // Row
        ],
      ); // Column // Container
    ],
  );
}
```

Exercice Application: Réaliser Cette page



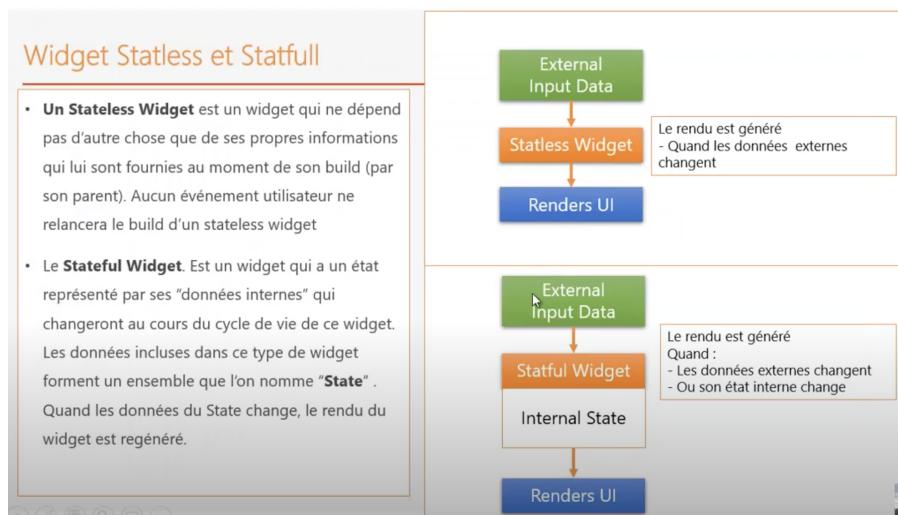
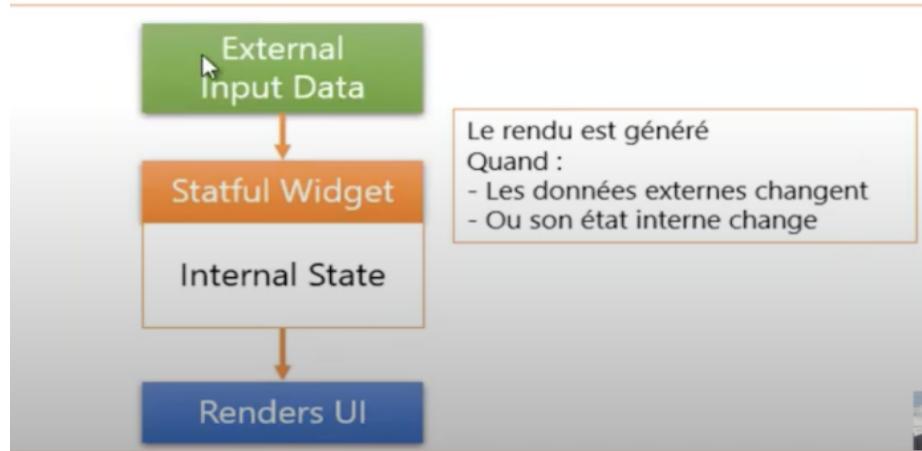
b. Les Interactifs ou Statefull Widget

Un **statefull widget** est un widget qui a un état représenté par des **données internes** qui peuvent changer au cours du cycle de vie de ce widget .

Les données incluses dans ce type de widget forment un ensemble qu'on appelle **state**. Lorsque les données du state changent , le rendu du widget est régénéré.

NB:

- **External Input Data :** Représente toutes données passées par le Parent Widget à son widget enfant lors de son build.
- **Internal State ou État :** signifie un ensemble de données dynamiques qui changent au cours de l'exécution de l'application.



Remarque

- Stateless widget ne change de rendu que si ses External Input Data changent.
- Stateful widget change de rendu lorsque ses External Input Data ou ses Internal State changent.

Quelques Exemples de Widget Interactif

1. Button

Exemple :Changer le background d'un **Scaffold** et le texte d'entête au clic d'un **floatActionButton**

a. Déclaration des variables dans la classe State

```
Color backgroundColor = Colors.white;  
  
Color TextColor = Colors.black;
```

b. Appliquer la variable au background du Scaffold

```
return Scaffold(  
  
    backgroundColor: backgroundColor,
```

c. Appliquer la variable au title du AppBar

```
appBar: AppBar(  
  
    title: Text(  
  
        "Widgets Interactifs",  
  
        style: TextStyle(color: TextColor),  
  
    ),
```

d. Changer l'état dans dans le FloatingActionButton

```
floatingActionButton: FloatingActionButton(
```

```
    onPressed: () {
```

```
        setState(() {
```

```
            backgroundColor =
```

```
(backgroundColor == Colors.white) ? Colors.black : Colors.white;
```

```
            TextColor =
```

```
(TextColor == Colors.white) ? Colors.black : Colors.white;
```

```
        });
```

```
    },
```

```
    child: Icon(Icons.build),
```

```
),
```

Exemple :Changer le contenu d'un texte au clic d'un TextButton

a. Déclaration des variables dans la classe State

```
bool isClicked = false;
```

```
String text = "Text Avant Clique";
```

b. Définition de la fonction de changement d'état

```
updateText() {
```

```
    setState(() {
```

```
        text = isClicked ? "Text Avant Clique" : "Mon Texte apres clique";
```

```
        isClicked = !isClicked;
```

```
    });
```

```
}
```

c. Appel de la fonction de changement d'état

```
body: Center(  
  
    child: Column(  
  
        mainAxisAlignment: MainAxisAlignment.center,  
  
        children: [  
  
            Text(text), TextButton(onPressed: updateText, child: Text("Mon Button"))  
  
        ],  
  
    ),  
  
),
```

Exemple :ElevatedButton

```
ElevatedButton(  
  
    onPressed: () {  
  
        print("Click Simple Button");  
  
    }),  
  
    onLongPress: () {  
  
        print("Click Long Button");  
  
    }),  
  
    style: ElevatedButton.styleFrom(  
  
        primary: Colors.amber,  
  
        elevation: 10,  
  
        shadowColor: Colors.black),  
  
    child: const Text("Evaluated Button"),  
)
```

Exemple : IconButton

```
IconButton(  
    onPressed: () {  
        print("J'ai cliquée sur mon IconButton");  
    }) ,  
    icon: Icon(Icons.favorite),  
    color: Colors.pink,  
    splashColor: Colors.pinkAccent,  
)
```

Exemple : TextField

a. Style

```
Container(  
    width: 200,  
    child: const TextField(  
        obscureText: false, //Cacher le Text Écrit  
        decoration: InputDecoration(  
            icon: Icon(Icons.send),  
            hintText: 'Hint Text', //PlaceHolder  
            helperText: 'Helper Text', //Text Helper  
            errorText: 'Error Text', //Text Erreur  
            counterText: '0 characters', //Nombre de Caractères  
            border: OutlineInputBorder(  
                borderRadius:  
                    const BorderRadius.all(Radius.circular(25))),
```

```
        ) ,  
  
        keyboardType: TextInputType.number, //Type du clavier ouvert  
  
    ) ,  
  
)
```

b. Interaction

```
onChanged: (value) {  
  
    setState(() {  
  
        text = value;  
  
    });  
  
},
```

Exemple: TextFieldEditingController

1. Déclaration du Controller

```
late TextEditingController _controller;
```

2. Initialisation du Controller

```
@override  
  
void initState() {  
  
super.initState();  
  
_controller = TextEditingController();  
  
}
```

3. Application du Controller au TextField

```
TextField(  
  
    onChanged: ((value) => setState(() {})),  
  
    decoration: const InputDecoration(
```

```
        icon: Icon(Icons.send),  
  
        hintText: 'Entrer le Prenom',  
  
    ),  
  
    keyboardType: TextInputType.number,  
  
    controller: _controller,  
  
    
```

4. Récupération de la valeur Controller

```
Text(_controller.text)
```

Exemple:Switch

1. Initialisation de la variable

```
late bool isSwitched;  
  
@override  
  
void initState() {  
  
    super.initState();  
  
    isSwitched = true;  
  
}
```

2. Définition du Switch

```
Text(isSwitched ? "J'ai clique" : "Je n'ai pas clique"),  
  
Switch(  
  
    inactiveTrackColor: Colors.red, //Couleur Inactive  
  
    activeColor: Colors.green, //Couleur Active  
  
    inactiveThumbColor: Colors.redAccent, //couleur de la zone  
circulaire  
  
    value: isSwitched,  
  
    onChanged: (value) {
```

```
        setState(() {  
  
            isSwitched = value;  
  
        });  
  
    }  
  
}
```

Exemple:Slider

```
double _currentSliderValue = 20;
```

```
Padding(  
  
    padding: const EdgeInsets.all(4.0),  
  
    child: Row(  
  
        children: [  
  
            Slider(  
  
                value: _currentSliderValue,  
  
                max: 100,  
  
                thumbColor: Colors.deepPurple, //couleur de la zone circulaire  
  
                inactiveColor: Colors.brown,  
  
                activeColor: Colors.amberAccent,  
  
                min: 0,  
  
                divisions: 5, //Pas  
  
                label: _currentSliderValue  
  
                    .round()  
  
                    .toString(), //affichage de la valeur selection  
  
                onChanged: (double value) {  
  
                    setState(() {  
  
                        _currentSliderValue = value;  
  
                    });  
  
                },  
  
            ),  
  
        ],  
  
    ),  
  
);
```

```
        } );  
  
    },  
  
)  
  
],  
  
) ,  
  
)
```

Exemple :Checkbox

1. Définition du Checkbox

```
bool isChecked = false;
```

```
Checkbox(  
  
        checkColor: Colors.white,  
  
        value: isChecked,  
  
        onChanged: (bool? value) {  
  
            setState(() {  
  
                isChecked = value!;  
  
            });  
  
        },  
  
        activeColor: Colors.white)
```

2. Générer des Checkbox dynamique

a. Définition du Map de langues

```
Map<String, bool> langues = {"Fr": true, "Eng": false, "Esp": false};
```

b. Fonction de Génération des checkbox

```
Column generateColumn() {  
  
    List<Row> rows = [];  
  
    langues.forEach((key, value) {  
  
        Row row = Row(  
  
            mainAxisAlignment: MainAxisAlignment.max,  
  
            mainAxisSize: MainAxisSize.max,  
  
            children: [  
  
                Text(key),  
  
                Checkbox(  
  
                    value: value,  
  
                    onChanged: (bool? newValue) {  
  
                        setState(() {  
  
                            langues[key] = newValue ?? false;  
  
                        });  
  
                    })  
  
            ]  
        );  
  
        rows.add(row);  
    });  
  
    return Column(  
  
        children: rows,  
    );  
}
```

c. Appel de la fonction

```
Padding(  
  padding: const EdgeInsets.only(left: 8.0),  
  child: generateColumn(),  
)
```

Exemple : Radio

Exemple 1: Utilisation sans enumeration

```
Row(  
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
  children: [  
    Text("Masculin"),  
    Radio(  
      value: 0, //Valeur initialisation  
      groupValue: radioValue, //Valeur Retournee  
      activeColor: Colors.amberAccent,  
      onChanged: (int? value) {  
        setState(() {  
          radioValue = value ?? 0;  
          print(radioValue);  
        });  
      },  
    ),  
    Text("Feminin"),
```

```
Radio(
```

```
    value: 1,
```

```
    groupValue: radioValue,
```

```
    onChanged: (int? value) {
```

```
        setState(() {
```

```
            radioValue = value ?? 0;
```

```
            print(radioValue);
```

```
        });
```

```
    },
```

```
),
```

```
],
```

```
),
```

Exemple 1: Utilisation avec enumeration

1. Définition de l'énumération

```
Map<String, bool> langues = {"Fr": true, "Eng": false, "Esp": false};  
  
Genre? _radioValue = Genre.Feminin;
```

```
Row(  
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
  
    children: [  
  
        Text("Masculin"),  
  
        Radio<Genre>(  
            value: Genre.Feminin, //Valeur initialisation  
  
            groupValue: _radioValue, //Valeur Retournee
```

```
        activeColor: Colors.amberAccent,
```

```
        onChanged: (Genre? value) {
```

```
            setState(() {
```

```
                _radioValue = value ?? Genre.Feminin;
```

```
            }) ;
```

```
        } ,
```

```
    ) ,
```

```
    Text("Feminin") ,
```

```
    Radio<Genre> (
```

```
        value: Genre.Masculin,
```

```
        groupValue: _radioValue,
```

```
        onChanged: (Genre? value) {
```

```
            setState(() {
```

```
                _radioValue = value;
```

```
            }) ;
```

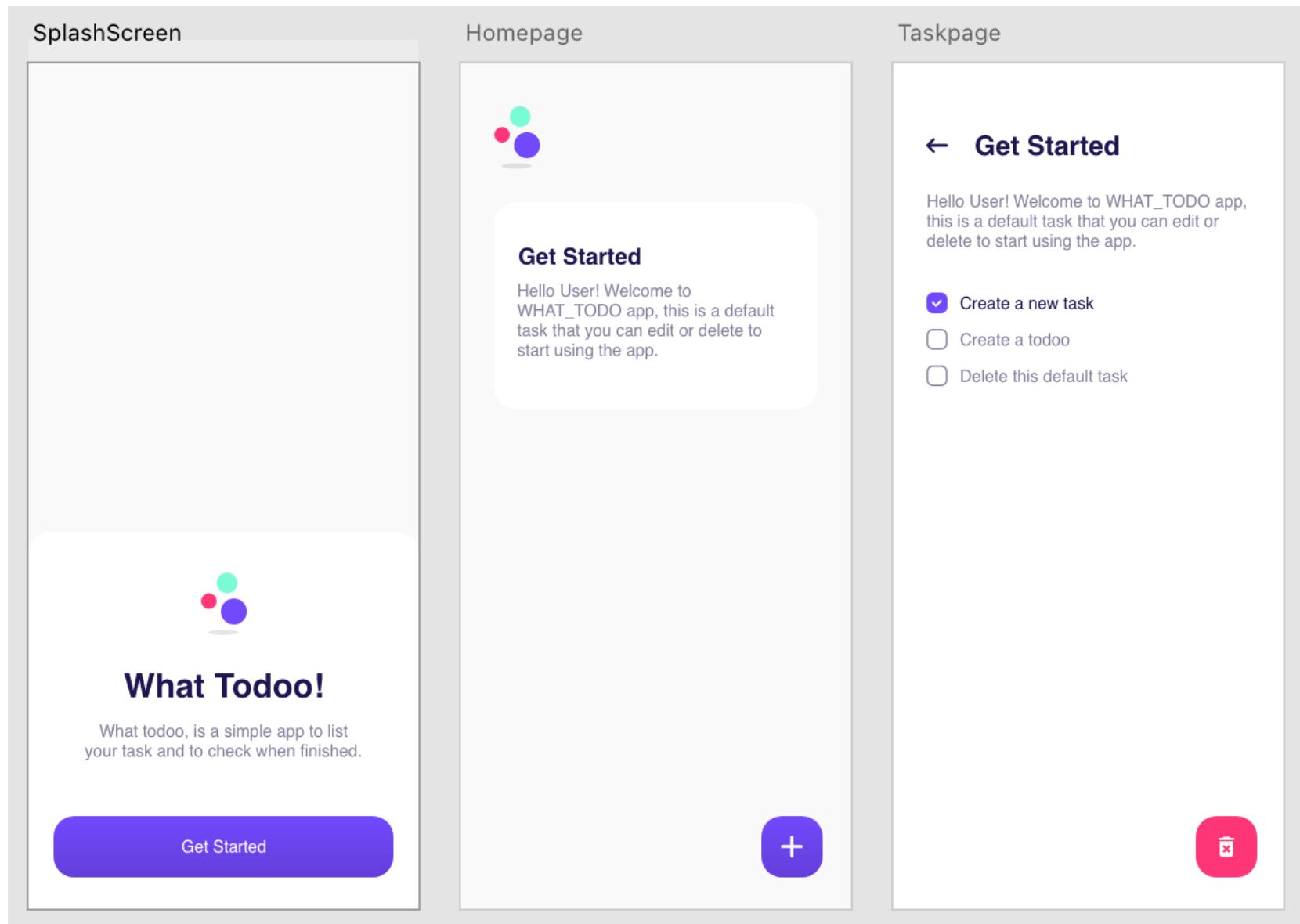
```
        } ,
```

```
    ) ,
```

```
    1 ,
```

```
),
```

APP 1 : Réalisation d'une Application de Todo List



NB : On peut aussi créer un projet en tapant la commande

1. Création du Projet

`flutter create todo_list`

2. Structure du dossier lib

- `models`: Définition des entités de l'application
- `pages`: Définition des pages de l'application
- `services`: Définition des fonctions d'accès aux données
- `widgets`: Définition des composants UI de notre projet

3. Gestion des assets

C. Création du dossier assets

d. Création du dossier images

e. Dans le fichier pubspec.yaml

```
# To add assets to your application, add an assets section, like this:  
assets:  
  - assets/images/
```

4. Widgets

i. SafeArea

SafeArea est un widget qui permet de rendre notre interface utilisateur adaptative et sans erreur en fonction de l'appareil qui l'exécute.

ii. ListView Widget

Listview en flutter est un widget utilisé pour afficher les éléments de manière linéaire. Ce widget permet de scroller sur l'axe horizontal ou vertical avec sa propriété **scrollDirection(Axis.Horizontal ou Axis.Vertical)**. On peut citer aussi d'autres propriétés telles que :

- **padding**
- **shrinkWrap(true or false)**: Permet de rendre flexible contenu de la liste
- **reverse(true or false)**: qui permet d'inverser la Liste
- **itemExtent (valeur double)**: qui fixe l'étendue d'un élément de liste

iii. ListView.builder()

builder() est une méthode static d'une **ListView**, utilisée pour générer dynamiquement les éléments de la liste ou avec des données de l'API (backend).

- NB: Ce constructeur convient si nous connaissons le nombre d'enfants car dans sa propriété **itemCount** on devra indiquer le nombre élément à construire.

iv. ListTitle

ListTitle est un widget utilisé pour afficher du texte et une icône, un sous titre ou un autre widget. On peut citer aussi d'autres propriétés telles que:

- **leading** : zone d'affichage de l'icône en début
- **title**: zone d'affichage d'un text comme titre
- **subtitle**: zone d'affichage d'un text comme sous titre
- **trailing** : zone d'affichage en fin d'un test, d'une icône ou tout autre widget
- **selected(true, false)**
- **enabled(true, false)**
- **dense(true, false)** : permet de réduire la taille du text
- **tileColor** : changer la couleur de fond
- **contentPadding**
-

v. Card

Card sont des zones affichages sous forme de bloc. Un card est généralement associé à un Site Title .

On peut citer aussi d'autres propriétés telles que:

- **elevation(double)**
 - **shadowColor**: couleur en ombre
shape : donner une form
- Exemple :
- ```
shape: BeveledRectangleBorder(
 borderRadius: BorderRadius.circular(15)
),
shape: OutlineInputBorder(
 borderRadius: BorderRadius.circular(10),
 borderSide: BorderSide(color: Colors.white)
),
shape: CircleBorder(side: BorderSide(width: 1, color: Colors.white))
```
- **margin:**
  - **height, width** : hauteur et la largeur

## 5. Page HomePage

### a. Code Couleur

- Couleur de Fond : #FFF6F6F6

### b. Intégration de la HomePage

#### i. Intégration du logo

```
return Scaffold(
 body: SafeArea(
 child: Container(
 width: double.infinity,
 padding: const EdgeInsets.symmetric(horizontal: 24.0, vertical: 32.0),
 color: Color(0xFFFF6F6F6),
 child: Column(
 crossAxisAlignment: CrossAxisAlignmentAlignment.start,
 children: [
 //Image(image: AssetImage("images/tasks/logo.png"))
 Container(
 margin: EdgeInsets.only(bottom: 22.0),
 child: Image.asset("assets/images/logo.png"),
),
],
),
),
);
```

NB : Pour Appliquer une couleur à partir d'une police, on utilise Colors(0xCodeCouleur) .

Exemple : `Color(0xFFFF6F6F6)`

#### ii. Widget Task

##### 1. Créer le widget TaskCard ( stateless)

##### 2. Code Couleur

- Couleur de Fond : white
- Couleur de Police
  - Texte entete: #FF211551
  - Texte Description: #FF866829D

### 3. Intégration du widget TaskCard ( stateless)

```
return Container(
 margin: EdgeInsets.only(bottom: 20.0),
 width: double.infinity,
 padding: EdgeInsets.symmetric(vertical: 32.0, horizontal: 24.0),
 decoration: BoxDecoration(
 color: Colors.white, borderRadius: BorderRadius.circular(20.0)),
 child: Column(
 mainAxisAlignment: MainAxisAlignment.start,
 children: const [
 Text(
 "Get Starting!!",
 style: TextStyle(
 color: Color(0xFF211551),
 fontSize: 22.0,
 fontWeight: FontWeight.w600),
),
 Padding(padding: EdgeInsets.only(top: 10.0)),
 Text(
 "Le Lorem Ipsum est simplement du faux texte employé dans la composition
faux texte standard",
 style: TextStyle(
 color: Color(0xFF866829D), fontSize: 16.0, height: 1.5),
),
],
);
}
```

### 4. Création du Model TaskCard

```
class TaskModel {
String title;
String description;
TaskModel({required this.title, required this.description});
}
```

## 5. Injection du model dans le Widget Task

```
final TaskModel task;
const TaskCardWidget({Key? key, required this.task}) : super(key: key);
```

## 6. Service Task

```
class TaskService {

 static List<TaskModel> getTasks() {

 return [

 TaskModel(
 title: "Ma Premiere Tache",
 description:
 "Le Lorem Ipsum est simplement du faux texte employé dans la
composition et la mise en page avant impression."),
 TaskModel(
 title: "Ma Deuxieme Tache",
 description:
 "Le Lorem Ipsum est simplement du faux texte employé dans la
composition et la mise en page avant impression.")
];
 }
}
```

## 7. Génération des Task Widget

```
List<TaskCardWidget> generateTaskCardWidget() {

 List<TaskCardWidget> children = [];
 List<TaskModel> tasks = TaskService.getTasks();
 tasks.forEach(((element) {
 children.add(TaskCardWidget(task: element));
 }));

 return children;
}
```

## 8. Ajout des Tasks dans une ListView

```
ListView(
 scrollDirection: Axis.vertical,
 children: generateTaskCardWidget(),
 shrinkWrap: true,
)
```

## 9. Ajout des Tasks dans une ListView.builder()

```
List<TaskModel> tasks = TaskService.getTasks();
Expanded(
 child: ListView.builder(
 itemBuilder: ((context, index) {
 return TaskCardWidget(task: tasks[index]);
 }),
 itemCount: tasks.length,
 scrollDirection: Axis.vertical,
),
```

## 10. Intégration du Bouton add

```
floatingActionButton: FloatingActionButton(
 backgroundColor: Color(0xFF7349FE),
 child: const Icon(
 Icons.add,
 color: Colors.white70,
),
 onPressed: () {},
) ,
```

## 11. Routing vers la Task Page

- a. Créer la page Task
- b. Faire le Routing de la Page Home vers la Page Task

```
onPressed: () {
 Navigator.push(
 context, MaterialPageRoute(builder: ((context) => TaskPage()));
},
```

## 12. Réalisation du Menu

### a. Création du Widget Menu

```
Drawer(
 backgroundColor: Color(0xFFFF6F6F),
 child: ListView(
 // Important: Remove any padding from the ListView.
 padding: EdgeInsets.zero,
 children: [
 Container(
 height: 80,
 child: const DrawerHeader(
 decoration: BoxDecoration(
 color: Colors.blue,
),
 child: Text(
 'Gestion des Tache',
 style: TextStyle(fontSize: 18.0, color: Colors.white),
),
),
),
 Card(
 margin: const EdgeInsets.all(6.0),
 child: ListTile(
 title: const Text(
 'Item 1',
 style: TextStyle(fontSize: 18.0),
),
 leading: const Icon(
 Icons.task_alt_rounded,
 color: Colors.blue,
),
 onTap: () {},
),
),
],
);
)
```

## b. Rendre Dynamique le Widget Menu

```
List<Card> generateMenuItem() {
 List datas = [
 {
 "title": "Liste des Taches",
 "icon": Icons.list_alt_outlined,
 "page": HomePage()
 },
 {"title": "Nouvelle Tache", "icon": Icons.add, "page": TaskPage()}
];
 List<Card> menus = [];
 datas.forEach((menu) {
 menus.add(Card(
 margin: const EdgeInsets.all(6.0),
 child: ListTile(
 title: Text(
 menu['title'],
 style: const TextStyle(fontSize: 18.0),
),
 leading: Icon(
 menu["icon"],
 color: Colors.blue,
),
 onTap: () {
 Navigator.push(context,
 MaterialPageRoute(builder: ((context) => menu["page"])));
 },
),
)));
 });
 return menus;
}
```

## 6. Page Task

### a. Intégration de la Page

## 7. Flutter et API REST

### a. Json-rest-server

#### i. Installation

1. Création d'un dossier **api**
2. Installation de **Json server**  
**npm install --save json-server**
3. créer le fichier **db.json**
4. initialiser le fichier **packadge.json**  
**npm init**

#### ii. Configuration

##### 1. configurer fichier packadge.json

```
"scripts": {
 "run-server": "json-server --host localhost db.json"
},
```

#### 2. Lancer le server

**npm run run-server**

#### iii. Création des ressources

```
{
 "tasks": [
 {
 "title": "Ma Premiere Tache",
 "description":
 "Le Lorem Ipsum est simplement du faux texte employé dans la composition
et la mise en page avant impression."
 },
 {
 "title": "Ma Deuxieme Tache",
 "description":
 "Le Lorem Ipsum est simplement du faux texte employé dans la composition
et la mise en page avant impression."
 },
 {
```

```

 "title": "Ma Troisieme Tache",
 "description":
 "Le Lorem Ipsum est simplement du faux texte employé dans la composition
et la mise en page avant impression."
},
{
 "title": "Ma Quatrieme Tache",
 "description":
 "Le Lorem Ipsum est simplement du faux texte employé dans la composition
et la mise en page avant impression."
},
{
 "title": "Ma Cinquieme Tache",
 "description":
 "Le Lorem Ipsum est simplement du faux texte employé dans la composition
et la mise en page avant impression."
}
],
}

```

## b. Intégration d'un api avec flutter

### i. Package

Pour intégrer un api Rest en flutter on peut utiliser deux packages :

1. Soit le package http

a. Installation :

```
flutter pub add http
```

b. Dans le Fichier pubspec.yaml

dependencies:

```
http: ^0.13.4
```

c. Importation du Package

```
import 'package:http/http.dart';
```

2. Soit le package Dio

a. Installation :

```
flutter pub add dio
```

b. Dans le Fichier pubspec.yaml

dependencies:

```
dio: ^4.0.6
```

c. Importation du Package

```
import 'package:dio/dio.dart';
```

NB : Dans ce cours nous utiliserons le package Dio

## ii. Consommation de l'api

# IV. SharedPreferences

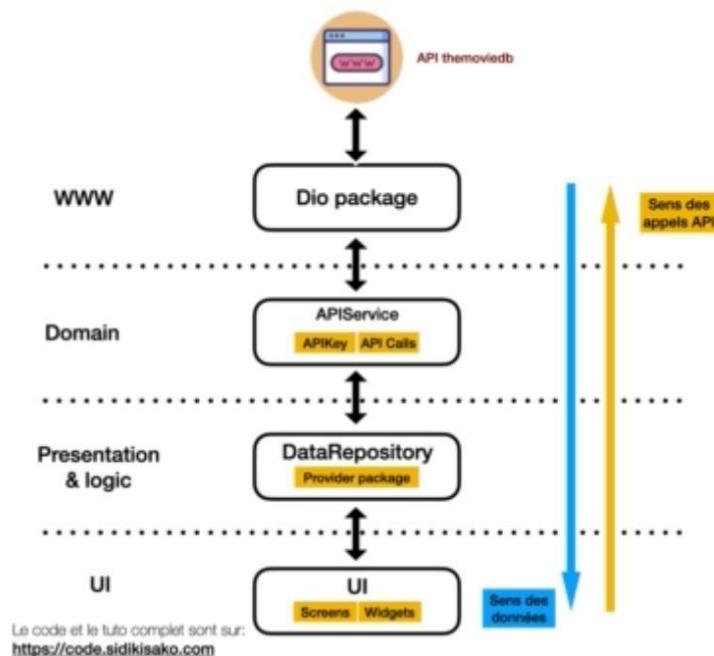
- A.** SharedPreferences-Obtenir des données
- B.** SharedPreferences-Ajouter des données
- C.** SharedPreferences-Supprimer des données

# APP 1 : Réalisation d'une Application de NetFlix

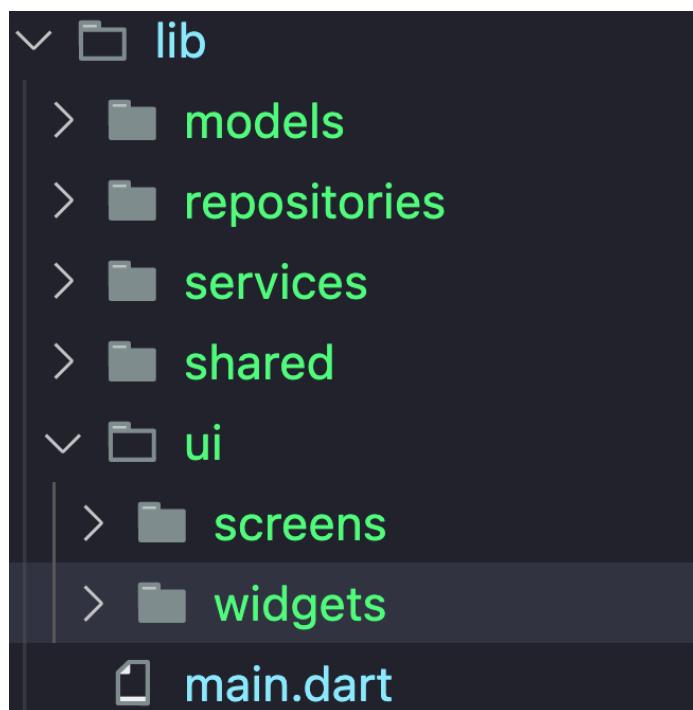
## 1. Gestion de API

- a. Créer un compte dans : <https://www.themoviedb.org/>
- b. Créer une clé Api

## 2. Architecture du Projet



## 3. Création du Projet



## **4. Ajouter le Projet sur GitHub**

# Création de la page Home

#ui/screen/home\_screen.dart

## a. Définition des constantes de couleur

#shared/constants.dart

```
Color bbwBackgroundColor = const Color(0xff020202);
Color bbwPrimaryColor = const Color(0xffe50914);
```

## b. AppBar

```
return Scaffold(
 backgroundColor: bbwBackgroundColor,
 appBar: AppBar(
 backgroundColor: bbwBackgroundColor,
 leading: Image.asset('assets/images/netflix_logo_2.png'),
)),
```

## c. Body

### i. Installation

#### 1. Installation de Google Font

##### a. Installation :

flutter pub add google\_fonts

##### b. Dans le Fichier pubspec.yaml

```
google_fonts: ^2.3.2
```

#### 2. Soit le package Dio

##### a. Installation :

flutter pub add dio

##### b. Dans le Fichier pubspec.yaml

dependencies:

dio: ^4.0.6

### ii. Widget du body

```
body: ListView(children: [])
```

### iii. Création des Widgets de la ListView du body

#### 1. Section Film à la une

```
Container(
 height: 500,
 color: bbwPrimaryColor,
)
```

#### 2. Section Tendances Actuelles

```
const SizedBox(
 height: 15.0,
) ,
Text(
 "Tendances actuelles",
 style: GoogleFonts.poppins(
 color: Colors.white, fontSize: 18.0, fontWeight: FontWeight.bold),
) ,
const SizedBox(
 height: 5.0,
) ,
SizedBox(
 height: 160,
 child: ListView.builder(
 scrollDirection: Axis.horizontal,
 itemCount: 10,
 itemBuilder: ((context, index) => Container(
 width: 110,
 margin: const EdgeInsets.only(right: 10),
 color: Colors.yellow,
 child: Center(child: Text(index.toString())),
)),
) ,
)
```

#### 3. Section Actuellement au cinéma

```
const SizedBox(
 height: 15.0,
) ,
Text(
 "Actuellement au Cinema",
 style: GoogleFonts.poppins(
 color: Colors.black, fontSize: 18.0, fontWeight: FontWeight.bold),
) ,
const SizedBox(
 height: 5.0,
```

```
) ,
 SizedBox(
 height: 320,
 child: ListView.builder(
 scrollDirection: Axis.horizontal,
 itemCount: 10,
 itemBuilder: ((context, index) => Container(
 width: 220,
 margin: const EdgeInsets.only(right: 10),
 color: Colors.blue,
 child: Center(child: Text(index.toString())),
)),
),
),
```

#### 4. Section Bientôt Disponible

```
const SizedBox(
 height: 15.0,
) ,
Text(
 "Bientot Disponible",
 style: GoogleFonts.poppins(
 color: Colors.black, fontSize: 18.0, fontWeight: FontWeight.bold),
) ,
const SizedBox(
 height: 5.0,
) ,
SizedBox(
 height: 160,
 child: ListView.builder(
 scrollDirection: Axis.horizontal,
 itemCount: 10,
 itemBuilder: ((context, index) => Container(
 width: 110,
 margin: const EdgeInsets.only(right: 10),
 color: Colors.green,
 child: Center(child: Text(index.toString())),
)),
)
```

#### iv. Création de la classe ApiKey

#services/api\_key.dart

```
class ApiKey {
 static String apiKey = "d0c94612323d062ddb29dda3ee387e31";
}
```

NB : ce fichier ne doit être ajouté dans le repository git

.gitignore

```
lib/services/api_key.dart
```

#### v. Création de la classe API

Définition des chemin de base

#services/api.dart

```
class API {
 final String apiKey = ApiKey.apiKey;
 final String baseURL = "https://api.themoviedb.org/3/";
 final String baseImageURL = "https://image.tmdb.org/t/p/w500/";
 final String baseVideoURL = "https://www.youtube.com/watch?v=";
}
```

#### vi. Création de la classe ApiService

#services/api\_service.dart

```
class ApiService {
 final API api = API();
 final Dio dio = Dio();

 Future<Response> getData(String path, {Map<String, dynamic>? params}) async {
 //On construit l'url
 String url = api.baseURL + path;
 //On construit l'url
 Map<String, dynamic> query = {'api_key': api.apiKey, 'language': "fr-FR"};
 if (params != null) {
 query.addAll(params);
 }
 final response = await dio.get(url, queryParameters: query);
 if (response.statusCode == 200) {
 return response;
 } else {
 throw Exception('Error: ${response.statusCode}');
 }
 }
}
```

```
 throw response;
 }
}
}
```

## vii. Création de la classe Model Movie

#models/model.dart

```
class Movie {
 final int id;
 final String name;
 final String description;
 final String? posterPath;
 Movie({
 required this.id,
 required this.name,
 required this.description,
 this.posterPath,
 }) ;

 Movie copyWith({
 int? id,
 String? name,
 String? description,
 String? posterPath,
 }) {
 return Movie(
 id: id ?? this.id,
 name: name ?? this.name,
 description: description ?? this.description,
 posterPath: posterPath ?? this.posterPath,
);
 }

 factory Movie.fromJson(Map<String, dynamic> map) {
 return Movie(
 id: map['id']?.toInt() ?? 0,
 name: map['title'] ?? '',
 description: map['overview'] ?? '',
 posterPath: map['poster_path'],
);
 }

 String posterURL() {
 API api = API();
 //! signifie qu'avant l'appel de cette fonction, on sera
 //assurée que le posterPath
```

```

 return api.baseImageURL + posterPath!;
 }
}

```

### viii. Rendre dynamique les Widgets du body

## Chargement de la section Tendance Actuelle

### 1. Dans le Service

#### #services/api\_service.dart

```

Future<List<Movie>> getPopularMovies({required int pageNumber}) async {
 Response response =
 await getData("/movie/popular", params: {'page': pageNumber});
 if (response.statusCode == 200) {
 Map data = response.data;
 List<dynamic> results = data['results'];
 List<Movie> movies = [];
 for (dynamic json in results) {
 Movie movie = Movie.fromJson(json);
 movies.add(movie);
 }
 return movies;
 } else {
 throw response;
 }
}

```

### 2. Récupération des films

#### #ui/screen/home\_screen.dart

```

class _HomeScreenState extends State<HomeScreen> {
 List<Movie>? movies;
 @override
 void initState() {
 // TODO: implement initState
 super.initState();
 getMovies();
 }

 void getMovies() {
 ApiService().getPopularMovies(pageNumber: 1).then((movieList) {
 setState(() {
 movies = movieList;
 });
 });
 }
}

```

```
)
```

### 3. Section Film à la une

```
Container(
 height: 500,
 color: bbwPrimaryColor,
 child: movies == null
 ? const Center()
 : Image.network(
 movies![0].posterURL(),
 fit: BoxFit.cover,
),
),
```

### 4. Section Tendances Actuelles

```
SizedBox(
 height: 160,
 child: ListView.builder(
 scrollDirection: Axis.horizontal,
 itemCount: 10,
 itemBuilder: ((context, index) => Container(
 width: 110,
 margin: const EdgeInsets.only(right: 10),
 color: Colors.yellow,
 child: movies == null
 ? Center(child: Text(index.toString()))
 : Image.network(
 movies![index].posterURL(),
 fit: BoxFit.cover,
),
))),
)
```

### 5. Mise en place des Repository

Les Services doivent être appelés par les repositories et pas par les screens.

```

#repositories/data_repository.dart
import 'package:dio/dio.dart';
import 'package:flutter/material.dart';
import 'package:netflex/services/api_service.dart';

import '../models/movie.dart';

class DataRepository with ChangeNotifier {
 final ApiService apiService = ApiService();
 final List<Movie> _popularMovieList = [];
 int _popularMovieIndex = 1;

 List<Movie> get popularMovieList => _popularMovieList;

 Future<void> getPopularMovies() async {
 try {
 List<Movie> movies =
 await apiService.getPopularMovies(pageNumber: _popularMovieIndex);
 _popularMovieList.addAll(movies);
 _popularMovieIndex++;
 notifyListeners();
 } on Response catch (response) {
 print("Error: ${response.statusCode}");
 rethrow;
 }
 }
}

```

**NB:** La méthode `notifyListeners();` vient de la classe `ChangeNotifier`, elle permet d' informer les abonnés de notre provider que des données de notre classe `DataProvider` ont changé.

## 6. Mise en place State Management

State Management permet accessibilité des données dans tous les screens.

### a. Installation de la dépendance

```
#pubspec.yaml
```

```
provider: ^6.0.2
```

**b. Rendre Accessible Provider dans toute l'application**

```
#main.dart
```

```
void main() {
 runApp(ChangeNotifierProvider(
 create: (context) => DataRepository(),
 child: const MyApp(),
));
}
```

**c. Utilisation du Provider**

```
#ui/screen/home_screen.dart
```

```
List<Movie>? movies;

void getMovies() async {
 final dataProvider = Provider.of<DataRepository>(context, listen: false);
 await dataProvider.getPopularMovies();
}
```

```
void initState() {
 // TODO: implement initState
 super.initState();
 getMovies();
}
```

**NB: Lors de linstanciation de la variable dataProvider dans le initState, on mettra le paramètre listen=false car le initState est exécuté quau chargement de la page.**

```
Widget build(BuildContext context) {
 final dataProvider = Provider.of<DataRepository>(context);
}
```

**NB: Les widgets qui utilisent la variable dataProvider sont des widgets abonnés au changement de notre classe DataRepository donc dans la déclaration de la variable dataProvider on ne mettra pas le paramètre listen=false;**

## 7. Creation Page Loading Screen

Les appels providers sont asynchrone peuvent mettre du temps pour s'exécuter, il faudra mettre un loader durant cette période d'appel.

### a. Installation de Dépendance

#pubspec.yaml

```
flutter_spinkit: ^5.1.0
```

### b. Création du widget de la page loading

#ui/screen/loading\_screen.dart

```
@override
Widget build(BuildContext context) {
 return Container(
 color: bbwBackgroundColor,
 child: Column(mainAxisAlignment: MainAxisAlignment.center, children: [
 Image.asset('assets/images/netflix_logo_1.png'),
 SpinKitFadingCircle(
 color: bbwPrimaryColor,
 size: 20,
),
]),
);
}
```

#main.dart

```
Widget build(BuildContext context) {
 return MaterialApp(
 title: 'My NetFlix',
 debugShowCheckedModeBanner: false,
 theme: ThemeData(
 primarySwatch: Colors.blue,
),
 home: LoadingScreen(),
);
}
```

### c. Gestion du State Management

#repositories/data\_repository.dart

```
Future<void> initData() async {
 await getPopularMovies();
}
```

#ui/screen/loading\_screen.dart

```
@override
void initState() {
 super.initState();
 initData();
}

Future<void> initData() async {
 final dataProvider = Provider.of<DataRepository>(context, listen: false);
 await dataProvider.initData();
 Navigator.pushReplacement(context, MaterialPageRoute(builder: (context) {
 return const HomeScreen();
 }));
}
```

#ui/screen/home\_screen.dart

### supprimer

```
void initState() {
 // TODO: implement initState
 super.initState();
 //getMovies();
}

/* void getMovies() async {
```

```
final dataProvider = Provider.of<DataRepository>(context, listen: false);
await dataProvider.getPopularMovies();
} */
```

## 8. Refactorisation du Code de HomeScreen

### a. Création des widgets

#### i. Widget CardMovie

Ce widget permet le chargement d'une image d'un film.

#installation de packadge pour la gestion des images network

#pubspec.yaml

```
cached_network_image: ^3.2.0
```

#Définition du widget

#ui/widget/movie\_card.widget

```
class MovieCard extends StatelessWidget {

 final Movie movie;

 const MovieCard({Key? key, required this.movie}) : super(key: key);

 @override

 Widget build(BuildContext context) {

 return CachedNetworkImage(

 imageUrl: movie.posterURL(),

 fit: BoxFit.cover,

 errorWidget: (context, url, error) =>

 const Center(child: Icon(Icons.error));
 }
 }
}
```

#Utilisation du widget dans HomePage

#ui/screen/home\_screen.dart

#code de Base

```
Image.network(
 dataProvider.popularMovieList[0].posterURL(),
 fit: BoxFit.cover,
) ,
```

#Remplacer par

```
MovieCard(movie: dataProvider.popularMovieList.first))
```

## ii. Widget CategorieMovie

Ce widget permet l'affichage d'une catégorie de film.

#Définition du widget

#ui/widget/movie\_categorie.widget

```
class MovieCategorie extends StatelessWidget {
 final String label;
 final List<Movie> movieList;
 final double imageHeight;
 final double imageWidth;
 const MovieCategorie(
 {Key? key,
 required this.movieList,
 required this.label,
 required this.imageHeight,
 required this.imageWidth})
 : super(key: key);

 @override
 Widget build(BuildContext context) {
 return Column(
 crossAxisAlignment: CrossAxisAlignment.start,
 children: [
 const SizedBox(
 height: 15.0,
),
 Text(
 label,
 style: GoogleFonts.poppins(
 color: Colors.white, fontSize: 18.0, fontWeight: FontWeight.bold),
),
 const SizedBox(
 height: 5.0,
),
 SizedBox(
 height: imageHeight,
 child: ListView.builder(
 scrollDirection: Axis.horizontal,
 itemCount: 10,
),
),
],
);
 }
}
extension MovieCategorieExtension on MovieCategorie {
 void showCategoryImage({required BuildContext context}) {
 Navigator.push(context, MaterialPageRoute(builder: (context) {
 return MovieCategorieView(movieList: movieList);
 }));
 }
}
```

```
 itemBuilder: ((context, index) => Container(
 width: imageWidth,
 margin: const EdgeInsets.only(right: 10),
 color: Colors.yellow,
 child: movieList.isEmpty
 ? Center(child: Text(index.toString()))
 : MovieCard(movie: movieList[index]),
))),
),
],
);
}
}
```

## #ui/screen/home\_screen.dart

### #code de Base

```
Container(
 height: 500,
 color: bbwPrimaryColor,
 child: dataProvider.popularMovieList.isEmpty
 ? const Center()
 : MovieCard(movie: dataProvider.popularMovieList.first)),
```

### #Remplacer Par

```
SizedBox(
 height: 500,
 child: MovieCard(movie: dataProvider.popularMovieList.first)),
```

### #code de Base

```
const SizedBox(
 height: 15.0,
),
Text(
 "Tendances actuelles",
 style: GoogleFonts.poppins(
 color: Colors.white, fontSize: 18.0, fontWeight: FontWeight.bold),
),
const SizedBox(
 height: 5.0,
),
SizedBox(
```

```

 height: 160,
 child: ListView.builder(
 scrollDirection: Axis.horizontal,
 itemCount: 10,
 itemBuilder: ((context, index) => Container(
 width: 110,
 margin: const EdgeInsets.only(right: 10),
 color: Colors.yellow,
 child: dataProvider.popularMovieList.isEmpty
 ? Center(child: Text(index.toString()))
 : MovieCard(
 movie: dataProvider.popularMovieList[index],
)));
),

```

## #Remplacer par

```

MovieCategorie(
 movieList: dataProvider.popularMovieList,
 label: "Tendances actuelles",
 imageHeight: 160,
 imageWidth: 110),

```

### iii. Pagination

Entourer la `ListView.builder` par `NotificationListener<ScrollNotification>` et activer l'événement `onNotification`.

`onNotification` est un callback qui est exécuté à chaque fois qu'on se déplace dans notre ListView.

Un Objet de type `ScrollNotification` contient les attributs

- `metrics.pixels` : qui retourne la position courante dans la liste view
- `metrics.maxScrollExten`: qui retourne la position courante dans la liste view

## #Widget CategorieMovie

```

class MovieCategorie extends StatelessWidget {
 final String label;
 final List<Movie> movieList;
 final double imageHeight;
 final double imageWidth;
 final Function callback;
 const MovieCategorie(
 {Key? key,
 required this.movieList,

```

```
 required this.label,
 required this.imageHeight,
 required this.imageWidth,
 required this.callback})
 : super(key: key);

}

@Override
Widget build(BuildContext context) {
 return Column(
 mainAxisAlignment: CrossAxisAlignment.start,
 children: [
 const SizedBox(
 height: 15.0,
),
 Text(
 label,
 style: GoogleFonts.poppins(
 color: Colors.white, fontSize: 18.0, fontWeight: FontWeight.bold),
),
 const SizedBox(
 height: 5.0,
),
 SizedBox(
 height: imageHeight,
 child: NotificationListener<ScrollNotification>(
 onNotification: (ScrollNotification notification) {
 final currentPosition = notification.metrics.pixels;
 final maxPosition = notification.metrics.maxScrollExtent;

 if (currentPosition >= maxPosition / 2) {
 callback();
 }
 return true;
 },
 child: ListView.builder(
 scrollDirection: Axis.horizontal,
 itemCount: movieList.length,
 itemBuilder: ((context, index) => Container(
 width: imageWidth,
 margin: const EdgeInsets.only(right: 10),
 child: movieList.isEmpty
 ? Center(child: Text(index.toString()))
 : MovieCard(movie: movieList[index]),
)));
],
);
}
```

```
) ,
] ,
)
};
}
}
```

## #ui/screen/home\_screen.dart

```
MovieCategorie(
 movieList: dataProvider.popularMovieList,
 label: "Tendances actuelles",
 imageHeight: 160,
 imageWidth: 110,
 callback: dataProvider.getPopularMovies,
) ,
```

## Chargement de la section les films à la une

### #services/api\_service.dart

```
Future<List<Movie>> getNowPlaying({required int pageNumber}) async {
 Response response =
 await getData("/movie/now_playing", params: {'page': pageNumber});
 if (response.statusCode == 200) {
 Map data = response.data;
 List<Movie> movies = data['results'].map<Movie>((dynamic json) {
 return Movie.fromJson(json);
 }).toList();
 return movies;
 } else {
 throw response;
 }
}
```

### #repositories/data\_repository.dart

```
//NowPlaying
final List<Movie> _nowPlaying = [];
int _nowPlayingPageIndex = 1;
List<Movie> get nowPlaying => _nowPlaying;
Future<void> getnowPlaying() async {
 try {
 List<Movie> movies =
 await apiService.getNowPlaying(pageNumber: _nowPlayingPageIndex);
 _nowPlaying.addAll(movies);
 _nowPlayingPageIndex++;
 notifyListeners();
 } on Response catch (response) {
 print("Error: ${response.statusCode}");
 rethrow;
 }
}
```

```
Future<void> initData() async {
 await getPopularMovies();
 await getnowPlaying();
}
```

### #ui/screen/home\_screen.dart

```
MovieCategorie(
 movieList: dataProvider.nowPlaying,
 label: "Actuellement au Cinéma",
 imageHeight: 360,
 imageWidth: 220,
```

```
callback: dataProvider.getnowPlaying,)
```

## Chargement de la section des films bientôt disponibles

### #services/api\_service.dart

```
Future<List<Movie>> getUpcomingMovies({required int pageNumber}) async {
 Response response =
 await getData("/movie/upcoming", params: {'page': pageNumber});
 if (response.statusCode == 200) {
 Map data = response.data;
 List<Movie> movies = data['results'].map<Movie>((dynamic json) {
 return Movie.fromJson(json);
 }).toList();
 return movies;
 } else {
 throw response;
 }
}
```

### #repositories/data\_repository.dart

```
final List<Movie> _upcomingMovies = [];
int _upcomingMoviesPageIndex = 1;
List<Movie> get upcomingMovies => _upcomingMovies;
Future<void> getUpcomingMovies() async {
 try {
 List<Movie> movies = await apiService.getUpcomingMovies(
 pageNumber: _upcomingMoviesPageIndex);
 _upcomingMovies.addAll(movies);
 _upcomingMoviesPageIndex++;
 notifyListeners();
 } on Response catch (response) {
 print("Error: ${response.statusCode}");
 rethrow;
 }
}
```

```
Future<void> initData() async {
 await getPopularMovies();
 await getnowPlaying();
 await getUpcomingMovies();
}
```

## #ui/screen/home\_screen.dart

```
MovieCategorie(
 movieList: dataProvider.upcomingMovies,
 label: "Bientot Disponible",
 imageHeight: 160,
 imageWidth: 110,
 callback: dataProvider.getUpcomingMovies,
),
```

## Section des films de la Catégorie Animation

### d. api :

- i. Section Genre :pour les genres de films
- ii. Section Discover : pour les films d'un genre

#### #services/api\_service.dart

```
Future<List<Movie>> getAnimationMovies({required int pageNumber}) async {
 Response response = await getData("/discover/movie",
 params: {'page': pageNumber, 'without_genres': '16'});
 if (response.statusCode == 200) {
 Map data = response.data;
 List<Movie> movies = data['results'].map<Movie>((dynamic json) {
 return Movie.fromJson(json);
 }).toList();
 return movies;
 } else {
 throw response;
 }
}
```

#### #repositories/data\_repository.dart

```
final List<Movie> _animationMovies = [];
int _animationMoviesPageIndex = 1;
List<Movie> get animationMovies => _animationMovies;
Future<void> getAnimationMovies() async {
 try {
 List<Movie> movies = await apiService.getAnimationMovies(
 pageNumber: _animationMoviesPageIndex);
 _animationMovies.addAll(movies);
 _animationMoviesPageIndex++;
 notifyListeners();
 } on Response catch (response) {
 print("Error: ${response.statusCode}");
 rethrow;
 }
}

Future<void> initData() async {
 await getPopularMovies();
 await getnowPlaying();
 await getUpcomingMovies();
 await getAnimationMovies();
}
```

## #ui/screen/home\_screen.dart

```
MovieCategorie(
 movieList: dataProvider.animationMovies,
 label: "Animations",
 imageHeight: 320,
 imageWidth: 220,
 callback: dataProvider.getAnimationMovies,
),
```

**NB : Dans la fonction initData() de Data Repository,**

```
Future<void> initData() async {
 await getPopularMovies();
 await getnowPlaying();
 await getUpcomingMovies();
 await getAnimationMovies();
}
```

Les appels des fonctions s'exécutent de manière séquentielle c'est-à-dire qu'il faudra que l'appel de la première fonction se termine pour qu'on puisse passer à l'appel de la fonction suivante et ainsi de suite.  
Cette approche n'est pas optimale du point de vue performance.

Pour le corriger, on fait des appels en parallèle

```
Future<void> initData() async {
 await Future.wait([
 getPopularMovies(),
 getnowPlaying(),
 getUpcomingMovies(),
 getAnimationMovies()
]);
}
```

# Page Détail d'un film

Api : Section Movies /Get Details

## 1. Mise à jour du Model Movie

```
class Movie {
 final int id;
 final String name;
 final String description;
 final String? posterPath;
 final List<String>? genres;
 final String? releaseDate;
 final double? vote;

 Movie(
 {required this.id,
 required this.name,
 required this.description,
 this.posterPath,
 this.genres,
 this.releaseDate,
 this.vote});

 Movie copyWith(
 {int? id,
 String? name,
 String? description,
 String? posterPath,
 List<String>? genres,
 String? releaseDate,
 double? vote}) {
 return Movie(
 id: id ?? this.id,
 name: name ?? this.name,
 description: description ?? this.description,
 posterPath: posterPath ?? this.posterPath,
 genres: genres ?? this.genres,
 releaseDate: releaseDate ?? this.releaseDate,
 vote: vote ?? this.vote,
);
 }

 factory Movie.fromJson(Map<String, dynamic> map) {
 return Movie(
 id: map['id']?.toInt() ?? 0,
 name: map['title'] ?? '',
 description: map['overview'] ?? '',
```

```

 posterPath: map['poster_path'] ,
) ;
}

String posterURL() {
 API api = API();
 /// signifie qu'avant l'appel de cette fonction, on sera
 //assuree que le posterPath
 return api.baseImageURL + posterPath!;
}

String reformatGenres() {
 return genres!.reduce((String value, String element) => '$value ,$element');
}
}

```

## 2. fonction getMoviesDetails() du service

```

Future<Movie> getMovieDetails({required Movie movie}) async {
 Response response = await getData(
 "/movie/${movie.id}",
);
 if (response.statusCode == 200) {
 Map data = response.data;
 var genres = data["genres"] as List;
 List<String> genreList =
 genres.map((genre) => genre['name'] as String).toList();
 Movie newMovie = movie.copyWith(
 genres: genreList,
 releaseDate: data["release_date"],
 vote: data['vote_average']);
 return newMovie;
 } else {
 throw response;
 }
}

```

## 3. fonction getMoviesDetail() du repository

```

Future<Movie> getMovieDetails({required Movie movie}) async {
 try {
 Movie newMovie = await apiService.getMovieDetails(movie: movie);
 return newMovie;
 } on Response catch (response) {
 print("Error: ${response.statusCode}");
 rethrow;
 }
}

```

## 4. Création de la Page Détail

### a. Création de Page MovieDetail

```
#ui/screen/movie_details.dart
import 'package:flutter/material.dart';
import 'package:flutter_spinkit/flutter_spinkit.dart';
import 'package:provider/provider.dart';

import '../../../../../models/movie.dart';
import '../../../../../repositories/data_repository.dart';
import '../../../../../shared/constants.dart';

class MovieDetailsPage extends StatefulWidget {
 final Movie movie;
 MovieDetailsPage({Key? key, required this.movie}) : super(key: key);

 @override
 State<MovieDetailsPage> createState() => _MovieDetailsPageState();
}

class _MovieDetailsPageState extends State<MovieDetailsPage> {
 Movie? newMovie;

 @override
 Widget build(BuildContext context) {
 return Scaffold(
 backgroundColor: bbwBackgroundColor,
 appBar: AppBar(
 backgroundColor: bbwBackgroundColor,
 leading: Image.asset('assets/images/netflix_logo_2.png'),
),
 body: newMovie == null
 ? Center(
 child: SpinKitFadingCircle(
 color: bbwPrimaryColor,
 size: 20,
),
)
 : ListView(
 children: [],
),
);
 }
}
```

### b. Modification de widget MovieCard

```
@override
Widget build(BuildContext context) {
 return GestureDetector(
 onTap: () {
 Navigator.push(
 context,
 MaterialPageRoute(
 builder: ((context) {
 return MovieDetailsPage(movie: movie);
 })),
);
 },
 child: CachedNetworkImage(
 imageUrl: movie.posterURL(),
 fit: BoxFit.cover,
 errorWidget: (context, url, error) =>
 const Center(child: Icon(Icons.error))),
);
}
```

### c. Crédit du widget MovieInfo

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

import '../../../../../models/movie.dart';

class MovieInfo extends StatelessWidget {
 final Movie newMovie;
 const MovieInfo({Key? key, required this.newMovie}) : super(key: key);

 @override
 Widget build(BuildContext context) {
 return Column(
 crossAxisAlignment: CrossAxisAlignment.start,
 children: [
 Container(
 height: 220,
 width: MediaQuery.of(context).size.width,
 color: Colors.red,
),
 const SizedBox(
 height: 10,
```

```
) ,
 Text(
 newMovie!.name,
 style: GoogleFonts.poppins(
 color: Colors.white, fontSize: 16.0, fontWeight: FontWeight.bold),
) ,
 const SizedBox(
 height: 5,
) ,
 Text(
 "Genre : ${newMovie!.reformatGenres()}" ,
 style: GoogleFonts.poppins(
 color: Colors.grey, fontSize: 14.0, fontWeight: FontWeight.w500),
 maxLines: 2,
 softWrap: true,
 overflow: TextOverflow.fade,
) ,
 const SizedBox(
 height: 5,
) ,
 Row(
 children: [
 Container(
 padding: const EdgeInsets.symmetric(vertical: 2, horizontal: 5),
 decoration: BoxDecoration(
 color: Colors.grey.withOpacity(0.3),
 borderRadius: BorderRadius.circular(5)),
 child: Text(
 newMovie!.releaseDate!.substring(0, 4),
 style: GoogleFonts.poppins(
 color: Colors.white,
 fontSize: 12,
 fontWeight: FontWeight.w500),
) ,
) ,
 const SizedBox(
 width: 15,
) ,
 Text(
 "Recommandé à ${(newMovie!.vote! * 10).toInt()}%",
 style: GoogleFonts.poppins(
 color: Colors.green,
 fontSize: 14.0,
 fontWeight: FontWeight.w500),
) ,
```

```
] ,
)
) ,
) ;
}
}
```

## d. Création des buttons Téléchargement et de Lecture

### i. Widget Action button

```
#ui/widgets/action_button.dart
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';

import '../../../../../shared/constants.dart';

class ActionButton extends StatelessWidget {
 final String label;
 final IconData icon;
 final Color bgColor;
 final Color color;
 const ActionButton(
 {Key? key,
 required this.label,
 required this.icon,
 required this.bgColor,
 required this.color})
 : super(key: key);

 @override
 Widget build(BuildContext context) {
 return Container(
 height: 50,
 width: MediaQuery.of(context).size.width,
 decoration:
 BoxDecoration(borderRadius: BorderRadius.circular(8), color: bgColor),
 child: Row(
 mainAxisAlignment: MainAxisAlignment.center,
 children: [
 Icon(
 icon,
 color: color,
),
 const SizedBox(
 height: 5,
),
],
),
);
 }
}
```

```

 Text(
 label,
 style: GoogleFonts.poppins(
 color: color, fontSize: 18.0, fontWeight: FontWeight.w500),
),
],
),
);
}
}

```

## ii. Movie Détails

#ui/screen/movie\_details.dart

```

class MovieDetailsPage extends StatefulWidget {
 final Movie movie;
 MovieDetailsPage({Key? key, required this.movie}) : super(key: key);

 @override
 State<MovieDetailsPage> createState() => _MovieDetailsPageState();
}

class _MovieDetailsPageState extends State<MovieDetailsPage> {
 Movie? newMovie;

 @override
 void initState() {
 // TODO: implement initState
 super.initState();
 getMovieData();
 }

 void getMovieData() async {
 final dataProvider = Provider.of<DataRepository>(context, listen: false);
 //Recupere les detail d'un movie
 Movie _movie = await dataProvider.getMovieDetails(movie: widget.movie);
 setState(() {
 newMovie = _movie;
 });
 }

 @override
 Widget build(BuildContext context) {
 return Scaffold(
 backgroundColor: bbwBackgroundColor,

```

```
appBar: AppBar(
 backgroundColor: bbwBackgroundColor,
 // leading: Image.asset('assets/images/netflix_logo_2.png'),
),
body: newMovie == null
? Center(
 child: SpinKitFadingCircle(
 color: bbwPrimaryColor,
 size: 20,
),
)
: Padding(
 padding: const EdgeInsets.all(8.0),
 child: ListView(
 children: [
 MovieInfo(newMovie: newMovie!),
 const SizedBox(
 height: 10,
),
 ActionButton(
 label: "Lecture",
 icon: Icons.play_arrow,
 bgColor: Colors.white,
 color: bbwBackgroundColor),
 const SizedBox(
 height: 5,
),
 ActionButton(
 label: "Telecharger la video",
 icon: Icons.download,
 bgColor: Colors.grey.withOpacity(0.3),
 color: Colors.white),
 const SizedBox(
 height: 20,
),
 Text(
 newMovie!.description,
 style: GoogleFonts.poppins(
 color: Colors.white,
 fontSize: 16.0,
),
),
],
),
),
```

```
) ;
}
}
```

### iii. Récupération et Affichage d'une vidéo

api: /movie/{movie\_id}/videos

#Dans le Fichier pubspec.yaml

```
youtube_player_flutter: ^8.1.0
```

```
#models/movie.dart
class Movie {

 final List<String>? videos;
```

```
 Movie(
 required this.id,
 required this.name,
 required this.description,
 this.posterPath,
 this.genres,
 this.releaseDate,
 this.vote,
 this.videos));
```

```
 Movie copyWith(
 {int? id,
 String? name,
 String? description,
 String? posterPath,
 List<String>? genres,
 String? releaseDate,
 double? vote,
 List<String>? videos}) {
 return Movie(
 id: id ?? this.id,
 name: name ?? this.name,
 description: description ?? this.description,
 posterPath: posterPath ?? this.posterPath,
 genres: genres ?? this.genres,
 releaseDate: releaseDate ?? this.releaseDate,
 vote: vote ?? this.vote,
 videos: videos ?? this.videos,
);
```

```
}
```

```
#services/api_service.dart
Future<Movie> getMovieVideos({required Movie movie}) async {
 Response response = await getData(
 "/movie/${movie.id}/videos",
);
 if (response.statusCode == 200) {
 Map data = response.data;

 List<String> videoKeys = data["results"]
 .map<String>((dynamic videoJson) => videoJson['key'] as String)
 .toList();
 Movie newMovie = movie.copyWith(videos: videoKeys);
 return newMovie;
 } else {
 throw response;
 }
}
```

```
#repositories/data_repository.dart
```

```
Future<Movie> getMovieDetails({required Movie movie}) async {
 try {
 //Recupere les infos Film
 Movie newMovie = await apiService.getMovieDetails(movie: movie);
 //Recupere les des videos
 newMovie = await apiService.getMovieVideos(movie: newMovie);
 return newMovie;
 } on Response catch (response) {
 print("Error: ${response.statusCode}");
 rethrow;
 }
}
```

```
#ui/widgets/my_video_player.dart
```

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

import '../../../../../shared/constants.dart';

class MyVideoPlayer extends StatefulWidget {
 final String movieId;
 const MyVideoPlayer({Key? key, required this.movieId}) : super(key: key);
```

```
@override
State<MyVideoPlayer> createState() => _MyVideoPlayerState();
}

class _MyVideoPlayerState extends State<MyVideoPlayer> {
YoutubePlayerController? _controller;
@Override
void initState() {
// TODO: implement initState
super.initState();
_controller = YoutubePlayerController(
initialVideoId: widget.movieId,
flags: const YoutubePlayerFlags(
mute: false,
autoPlay: false,
hideThumbnail: true,
),
);
}
}

@Override
void dispose() {
_controller!.dispose();
// TODO: implement dispose
super.dispose();
}

@Override
Widget build(BuildContext context) {
return _controller == null
? Center(
child: SpinKitFadingCircle(
color: bbwPrimaryColor,
size: 20,
),
)
: YoutubePlayer(controller: _controller!);
}
}
```

**NB :La classe Youtube Player offre des attributs tels que:**

- `progressColors` qui offre aussi les attributs et de callback
  - Les attributs
    - `handleColor`: couleur du bouton de progression de la vidéo
    - `playedColor` :couleur de la partie lue de la vidéo

```
YoutubePlayer(
 controller: _controller!,
 progressColors: ProgressBarColors(
 handleColor: bbwPrimaryColor,
 playedColor: bbwPrimaryColor,
),
 onPressed: (YoutubeMetaData meta) {
 _controller!.play();
 _controller!.pause();
 },
);
```

**Le callback `onEnd` de la classe `YoutubePlayer` permet de définir des actions à la fin de la vidéo**

```
onEnded: (YoutubeMetaData meta) {
 _controller!.play();
 _controller!.pause();
},
```

```
#ui/widgets/movie_info.dart

@Override
Widget build(BuildContext context) {
 return Column(
 crossAxisAlignment: CrossAxisAlignment.start,
 children: [
Container(
 height: 220,
 width: MediaQuery.of(context).size.width,
 child: newMovie.videos!.isEmpty
 ? Center(
 child: Text(
 "Pas de Video",
 style: GoogleFonts.poppins(
 color: Colors.white,
 fontSize: 16.0,
 fontWeight: FontWeight.bold),
),
)
 : MyVideoPlayer(movieId: newMovie.videos!.first)

),
//-----
// -----
);
}
}
```

```
#ui/screen/movie_details.dart
return Scaffold(
 backgroundColor: bbwBackgroundColor,
 appBar: AppBar(
 backgroundColor: bbwBackgroundColor,
 // leading: Image.asset('assets/images/netflix_logo_2.png'),
),
 body: newMovie == null
 ? Center(
 child: SpinKitFadingCircle(
 color: bbwPrimaryColor,
 size: 20,
),
)
);
```

```
)
: Padding(
 padding: const EdgeInsets.all(8.0),
 child: ListView(
 children: [
 MovieInfo(newMovie: newMovie!),
 const SizedBox(
 height: 10,
),
 ActionButton(
 label: "Lecture",
 icon: Icons.play_arrow,
 bgColor: Colors.white,
 color: bbwBackgroundColor),
 const SizedBox(
 height: 5,
),
 ActionButton(
 label: "Telecharger la video",
 icon: Icons.download,
 bgColor: Colors.grey.withOpacity(0.3),
 color: Colors.white),
 const SizedBox(
 height: 20,
),
 Text(
 newMovie!.description,
 style: GoogleFonts.poppins(
 color: Colors.white,
 fontSize: 16.0,
),
),
],
),
);
};
```

#### iv. Récupération et Affichage des acteurs

api: /movie/{movie\_id}/credits

```
#models/person.dart
class Person {
 final String name;
 final String characterName;
 final String? imageURL;
 Person({
 required this.name,
 required this.characterName,
 this.imageURL,
 });

 Person copyWith({
 String? name,
 String? characterName,
 String? imageURL,
 }) {
 return Person(
 name: name ?? this.name,
 characterName: characterName ?? this.characterName,
 imageURL: imageURL ?? this.imageURL,
);
 }

 //Renommer fromMap to FromJson
 factory Person.fromJson(Map<String, dynamic> map) {
 return Person(
 name: map['name'] ?? '',
 characterName: map['character'] ?? '',
 imageURL: map['profile_path'],
);
 }
}
```

```
#models/movie.dart
class Movie {
 final int id;
 final String name;
 final String description;
 final String? posterPath;
 final List<String>? genres;
 final String? releaseDate;

 final double? vote;
}

#models/person.dart
final List<String>? videos;
final List<Person>? casting;

Movie(
 required this.id,
 required this.name,
 required this.description,
 this.posterPath,
 this.genres,
 this.releaseDate,
 this.vote,
 this.videos,
 this.casting);

Movie copyWith(
 {int? id,
 String? name,
 String? description,
 String? posterPath,
 List<String>? genres,
 String? releaseDate,
 double? vote,
 List<String>? videos,
 List<Person>? casting}) {
 return Movie(
 id: id ?? this.id,
 name: name ?? this.name,
 description: description ?? this.description,
 posterPath: posterPath ?? this.posterPath,
 genres: genres ?? this.genres,
 releaseDate: releaseDate ?? this.releaseDate,
 vote: vote ?? this.vote,
 videos: videos ?? this.videos,
 casting: casting ?? this.casting,
);
}
```

```
) ;
}

#services/api_service.dart
Future<Movie> getMovieCast({required Movie movie}) async {
 Response response = await getData(
 "/movie/${movie.id}/credits",
);
 if (response.statusCode == 200) {
 Map data = response.data;

 List<Person> casting = data["cast"]
 .map<Person>((dynamic personJson) => Person.fromJson(personJson))
 .toList();
 Movie newMovie = movie.copyWith(casting: casting);
 return newMovie;
 } else {
 throw response;
 }
}
```

```
#repositories/data_repository.dart
Future<Movie> getMovieDetails({required Movie movie}) async {
 try {
 //Recupere les infos Film
 Movie newMovie = await apiService.getMovieDetails(movie: movie);
 //Recupere les des videos
 newMovie = await apiService.getMovieVideos(movie: newMovie);
 //Recuperer le casting
 newMovie = await apiService.getMovieCast(movie: newMovie);
 return newMovie;
 } on Response catch (response) {
 print("Error: ${response.statusCode}");
 rethrow;
 }
}
```

```
#ui/widgets/casting.dart

class CastingCard extends StatelessWidget {
 final Person person;
 const CastingCard({Key? key, required this.person}) : super(key: key);

 @override
```

```
Widget build(BuildContext context) {
 return Card(
 shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(5)),
 child: Column(
 children: [
 ClipRRect(
 borderRadius: BorderRadius.circular(5),
 child: CachedNetworkImage(
 width: 170,
 imageUrl: person.imagePerson(),
 fit: BoxFit.cover,
 errorWidget: (context, url, error) =>
 const Center(child: Icon(Icons.error)),
),
),
 const Padding(padding: EdgeInsets.only(top: 10, left: 10)),
 SizedBox(
 width: 130,
 child: Text(person.name,
 style: GoogleFonts.poppins(
 color: Colors.black,
 fontSize: 15,
 fontWeight: FontWeight.bold)),
),
 const Padding(padding: EdgeInsets.only(left: 10)),
 SizedBox(
 width: 130,
 child: Text(person.characterName,
 style: GoogleFonts.poppins(
 color: Colors.black,
 fontSize: 14,
)),
),
],
),
);
}
```

```

#ui/screen/movie_details.dart
Widget build(BuildContext context) {
 return Scaffold(
 const SizedBox(
 height: 20,
),
 Text(
 "Casting",
 style: GoogleFonts.poppins(
 color: Colors.white,
 fontSize: 16.0,
 fontWeight: FontWeight.bold),
),
 SizedBox(
 height: 350,
 child: ListView.builder(
 itemCount: newMovie!.casting!.length,
 scrollDirection: Axis.horizontal,
 itemBuilder: (context, int index) {
 return newMovie!.casting![index] == null
 ? const Center()
 : CastingCard(person: newMovie!.casting![index]);
 },
),
),
],
),
);
}

```

## v. Affichage des images

```

api: /movie/{movie_id}/images
include_image_language=en,null
#models/movie.dart
final List<String>? images;

Movie(
 {required this.id,
 required this.name,
 required this.description,
 this.posterPath,
 this.genres,

```

```

 this.releaseDate,
 this.vote,
 this.videos,
 this.casting,
 this.images));
}

Movie copyWith(
 {int? id,
 String? name,
 String? description,
 String? posterPath,
 List<String>? genres,
 String? releaseDate,
 double? vote,
 List<String>? videos,
 List<Person>? casting,
 List<String>? images}) {
 return Movie(
 id: id ?? this.id,
 name: name ?? this.name,
 description: description ?? this.description,
 posterPath: posterPath ?? this.posterPath,
 genres: genres ?? this.genres,
 releaseDate: releaseDate ?? this.releaseDate,
 vote: vote ?? this.vote,
 videos: videos ?? this.videos,
 casting: casting ?? this.casting,
 images: images ?? this.images,
);
}

```

```

#services/api_service.dart
Future<Movie> getMovieImage({required Movie movie}) async {
 Response response = await getData("/movie/${movie.id}/images",
 params: {'include_image_language': "null"});
 if (response.statusCode == 200) {
 Map data = response.data;

 List<String> images = data["backdrops"]
 .map<String>((dynamic imageJson) =>
 "${api.base imageURL}${imageJson['file_path']}")
 .toList();
 Movie newMovie = movie.copyWith(images: images);
 return newMovie;
 } else {

```

```
 throw response;
 }
}
```

```
#repositories/data_repository.dart
Future<Movie> getMovieDetails({required Movie movie}) async {
 try {
 //Recupere les infos Film
 Movie newMovie = await apiService.getMovieDetails(movie: movie);
 //Recupere les des videos
 newMovie = await apiService.getMovieVideos(movie: newMovie);
 //Recuperer le casting
 newMovie = await apiService.getMovieCast(movie: newMovie);
 //Recuperer les Images
 newMovie = await apiService.getMovieImage(movie: newMovie);
 return newMovie;
 } on Response catch (response) {
 print("Error: ${response.statusCode}");
 rethrow;
 }
}
```

```
#ui/widget/image_card.dart
```

```
class ImageCard extends StatelessWidget {
 final String image;
 const ImageCard({Key? key, required this.image}) : super(key: key);

 @override
 Widget build(BuildContext context) {
 return Card(
 shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(5)),
 child: SizedBox(
 child: ClipRRect(
 borderRadius: BorderRadius.circular(5),
 child: CachedNetworkImage(
 width: 170,
 imageUrl: image,
 fit: BoxFit.cover,
 errorWidget: (context, url, error) =>
 const Center(child: Icon(Icons.error)),
),
),
),
);
 }
}
```

```
) ;
}
}
}
```

#ui/screen/movie\_detail.dart

```
SizedBox(
 height: 200,
 child: ListView.builder(
 itemCount: newMovie!.images!.length,
 scrollDirection: Axis.horizontal,
 itemBuilder: (context, int index) {
 return newMovie!.images![index] == null
 ? const Center()
 : ImageCard(image: newMovie!.images![index]);
 },
),
,
```

# Amélioration des Performances

Documentation :

[Getting Started](#)>[Append to Response](#)

#repositories/data\_repository.dart

Optimisation des appels dans la fonction `getMovieDetail()`

```
Future<Movie> getMovieDetails({required Movie movie}) async {
 try {
 //Recupere les infos Film
 Movie newMovie = await apiService.getMovieDetails(movie: movie);
 //Recupere les des videos
 newMovie = await apiService.getMovieVideos(movie: newMovie);
 //Recuperer le casting
 newMovie = await apiService.getMovieCast(movie: newMovie);
 //Recuperer les Images
 newMovie = await apiService.getMovieImage(movie: newMovie);
 return newMovie;
 } on Response catch (response) {
 print("Error: ${response.statusCode}");
 rethrow;
 }
}
```

## Correction

#services/api\_service.dart

```
Future<Movie> getMovie({required Movie movie}) async {
 Response response = await getData("/movie/${movie.id}", params: {
 'include_image_language': "null",
 'append_to_response': "videos,images,credits"
 });
 if (response.statusCode == 200) {
 Map data = response.data;
 //On recupere les genre
 var genres = data["genres"] as List;
 List<String> genreList =
 genres.map((genre) => genre['name'] as String).toList();
 //On recupere les videos
 List<String> videoKeys = data["videos"]["results"]
 .map<String>((dynamic videoJson) => videoJson['key'] as String)
 .toList();
 //On recupere les Photos
 List<String> images = data["images"]["backdrops"]
 .map<String>((dynamic imageJson) =>
```

```
"${api.baseImageURL}${imageJson['file_path']}")
 .toList();
//On recupere le Casting
List<Person> casting = data["credits"]["cast"]
 .map<Person>((dynamic personJson) => Person.fromJson(personJson))
 .toList();
Movie newMovie = movie.copyWith(
 genres: genreList,
 releaseDate: data["release_date"],
 vote: data['vote_average'],
 videos: videoKeys,
 images: images,
 casting: casting);
return newMovie;
} else {
 throw response;
}
}
```

```
#repositories/data_repository.dart
Future<Movie> getMovieDetails({required Movie movie}) async {
 try {
 //Recupere les infos Film
 Movie newMovie = await apiService.getMovie(movie: movie);
 return newMovie;
 } on Response catch (response) {
 print("Error: ${response.statusCode}");
 rethrow;
 }
}
```