|  |  |
| --- | --- |
| Flutter |  |
| **Fiche 1 : layout & stateless widgets** | |

Table des matières

[1 Objectifs à valider 2](#_Toc127461049)

[2 Installation de l’environnement 2](#_Toc127461050)

[**3** Mise en place de votre GitHub repository pour les exercices 2](#_Toc127461051)

[4 Introduction au framework Flutter 3](#_Toc127461052)

[5 Introduction au langage Dart 4](#_Toc127461053)

[6 Concepts 4](#_Toc127461054)

[6.1 Introduction 4](#_Toc127461055)

[6.2 Création du code de base d’une app 4](#_Toc127461056)

[6.3 Modélisation possible du layout d’un écran 5](#_Toc127461057)

[6.4 Création de layout et de stateless widgets de base 5](#_Toc127461058)

[6.5 Ajout d’une image dans votre app 6](#_Toc127461059)

[6.6 Création d’un stateless widget réutilisable & librairies 7](#_Toc127461060)

[6.7 Création et paramétrisation d’un stateless widget 10](#_Toc127461061)

[6.8 Comprendre les contraintes 13](#_Toc127461062)

[7 Exercice 14](#_Toc127461063)

[7.1 Introduction 14](#_Toc127461064)

[7.2 HomeScreen via un stateless widget 14](#_Toc127461065)

[7.3 Création d’un stateless widget paramétrable 15](#_Toc127461066)

[7.4 Peaufinage du layout 15](#_Toc127461067)

[7.5 Challenge optionnel 16](#_Toc127461068)

# Objectifs à valider

|  |  |  |
| --- | --- | --- |
| **ID** | **Objectifs** | **Indicateurs** |
| F01 | Création d’un stateless widget | * Les messages de commit F01.1, F01.2 & F01.3 sont visibles. * Vous montrez et expliquez l’utilisation d’un stateless widget composé lui-même de plusieurs widgets. * Vous justifiez comment vous avez paramétré un stateless widget. * Vous montrez un exemple de positional argument et un exemple de named arguments dans votre code. * Vous justifiez vos choix de layout pour mettre en valeur les informations de votre app, principalement par rapport à l’espace entre le titre de l’écran et la motivation ; présence d’une couleur différente pour un texte. |

# Installation de l’environnement

Si vous utilisez votre propre machine et que ça n’est pas déjà fait, rendez-vous sur la page « Get started » de Flutter : <https://flutter.dev/docs/get-started/install>

Et faites les points 1 à 3 compris (Test drive).

NB : lors du Test drive, il est probable que vous ayez ce « problème » identifié par **flutter doctor** :

[X] Visual Studio - develop for Windows

Nous vous proposons de ne pas résoudre ce « problème ». En effet, il n’est pas nécessaire de déployer nos applications Mobile en tant qu’application de bureau Windows.

En tant qu’IDE, nous recommandons l’utilisation d’Android Studio.

Pour les machines de l’école, vous devez mettre à jour flutter ainsi qu’Android Studio. Pour la mise à jour de flutter, ouvrez une console et tapez : **flutter upgrade**

NB : Si vous avez ce souci identifié par **flutter doctor :**

[!] Android Studio (version 2021.2)

X Unable to find bundled Java version.

La résolution est offerte sur [stack overflow](https://stackoverflow.com/questions/75115909/flutter-android-studio-version-2022-1-x-unable-to-find-bundled-java-versio/75125986).

|  |
| --- |
| Mise en place de votre GitHub repository pour les exercices |

Pour la vérification de vos objectifs à valider, nous vous demandons de travailler avec GitHub, via GitHub Classroom, pour versionner les projets que vous allez créer.

Veuillez accéder à l’assignement du cours de Mobile sur GitHub Classroom : <https://classroom.github.com/a/6eLS5d_Z>.

Loggez-vous au sein de GitHub via votre compte puis cliquez sur « Accept this assignment ». Raffraichissez votre page pour obtenir l’URL de votre web repo, par exemple <https://github.com/e-vinci/mobile-2023-exercises-e-baron>.

Cliquez sur l’URL de votre web repo. GitHub vous offre les instructions pour créer un nouveau repo local et le synchroniser avec votre GitHub repo.

Créer un dossier « **mobile** » (ou donnez-lui le nom que vous voulez) sur votre machine à un endroit ne se trouvant pas sur un « cloud drive » comme OneDrive, Google Drive ou autre.

C’est ce dossier que vous allez transformer en repo local en le liant à votre GitHub repo. Entrez dans ce dossier à l’aide de Git Bash. Vous pouvez ensuite tapez les commandes offertes par votre GitHub repo sous « …or create a new repository on the command line ».

Pour chaque exercice, faites attention à créer un nouveau projet au sein d’un dossier dans ce repository (se trouvant dans votre dossier « **mobile** » ou autre) sur votre machine. Vous pourrez de cette façon utiliser l’interface de Android studio pour effectuer des commits et des push. Nous vous demanderons à plusieurs moments au sein des fiches du cours de faire des commits sur ce repository avec un tag **[Commit "message de commit"]**. Cela nous permettra de vérifier l’état de votre avancement dans la matière du cours, et votre implication.

Pour cette raison, nous vous demandons de bien vouloir suivre les messages de commits demandés. Vous pouvez faire d’autres commits intermédiaires si vous le souhaitez, mais nous nous attendons à retrouver ceux-là au minimum. Faites particulièrement attention aux commits correspondant à des objectifs du cours, dont le message commence par FXY. Nous nous baserons sur le contenu de ces commits afin de valider si vous avez atteint les différents objectifs du cours.

# Introduction au framework Flutter

Pour avoir une introduction à Flutter, veuillez :

1. Visionnez cette vidéo d’introduction (4 minutes) : [How is flutter different for app devlopment](https://www.youtube.com/watch?v=l-YO9CmaSUM).
2. Vous rendre sur la homepage du Framework : <https://flutter.dev/> & lire les explications sur les objectifs (Fast, Productive, Flexible…) en cliquant notamment sur « Productive » et « Flexible ». N’hésitez pas à « jouer » quelques minutes (bouton « Try it in DartPad ») avec l’application exécutable dans votre browser.
3. Pour en savoir plus sur la philosophie de Design des UI dans Flutter, vous pouvez lire la (courte) page suivante : <https://flutter.dev/docs/get-started/flutter-for/declarative>

# Introduction au langage Dart

Nous vous avons présenté notre Dart CheatSheet. Celle-ci se trouve sur Moodle.

Toutefois si vous voulez avoir une vue plus détaillée du langage avant de commencer ou si durant la fiche vous aimeriez en savoir plus sur certaines constructions, rendez-vous sur le [language-tour](https://dart.dev/guides/language/language-tour).

# Concepts

## Introduction

Comme pour beaucoup de technologies de Google, une série de tutoriaux introductifs de qualités sont proposés. Pour attaquer cette technologie, vous allez commencer par effectuer plusieurs de ceux-ci. À la suite de cela, pour valider votre maîtrise de Flutter, nous vous demanderons de réaliser un exercice.

**⚡ Avertissement** : il est tentant de faire les tutoriaux à grande vitesse en faisant des copier-coller du code proposé par Google. Pour vraiment profiter d’un tutoriel comme outil d’apprentissage, il faut les faire en tant qu’acteur réflexif. A chaque bout de code que vous ajoutez, vous devez vous poser la question de sa raison d’être, des mécanismes qu’il met en œuvre etc. A certains moments nous vous aiderons dans ce processus d’assimilation en vous posant explicitement des questions. Mais cela ne sera pas toujours le cas. Veuillez donc à rester acteur de vos apprentissages 😉.

Nous allons maintenant commencer ce [tutoriel](https://docs.flutter.dev/development/ui/layout/tutorial)online.   
Veuillez lire l’introduction et revenir dans la fiche avant de passer au Step 0 du tutoriel online.

## Création du code de base d’une app

**Pour commencer le tutoriel, créez un nouveau projet (New Flutter Project…**, sélectionnez **Flutter**, puis **Project Type : Application) nommé *ftuto1* dans votre repository de cours.**

**Vous pouvez laisser sélectionné toutes les plateformes avant de cliquer sur *Create*.**

Si vous avez un téléphone à disposition ou un ordinateur suffisamment puissant pour exécuter un **Virtual Device**, vous pouvez exécuter l’application sur celui-ci.

Pour toutes les prochaines exécutions de vos programmes lors de la phase de développement, nous vous proposons de le faire pour la plateforme **Web**, via votre browser favori (Chrome par exemple). Ce type d’exécution consomme nettement moins de ressource de votre machine.

Graphical user interface, application

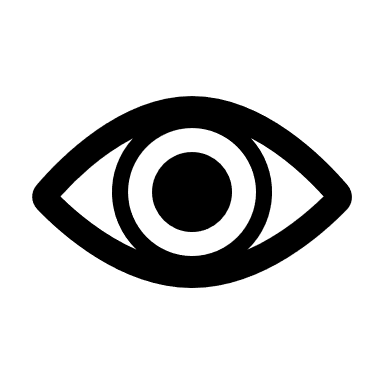
Description automatically generated

**[commit** **avec message : FT01.1 starter app]**

Remplacez le contenu de **lib/main.dart** par :

import 'package:flutter/material.dart';  
  
void main() {  
 runApp(const MyApp());  
}  
  
class MyApp extends StatelessWidget {  
 const MyApp({Key? key}) : super(key: key);  
  
 @override  
 Widget build(BuildContext context) {  
 return MaterialApp(  
 title: 'Welcome to Flutter',  
 home: Scaffold(  
 appBar: AppBar(  
 title: const Text('Welcome to Flutter'),  
 ),  
 body: const Center(  
 child: Text('Hello World'),  
 ),  
 ),  
 );  
 }  
}

Pour prendre en compte le nouveau code, cliquez sur le bouton de **Hot Reload** (l’éclair) : 

****Observations & questions

* Cet exemple crée une **Material** app. [Material](https://m2.material.io/design" \t "_blank) est un système de conception visuelle qui est standard pour le Mobile et le Web. Flutter offre beaucoup de Material widgets.   
  NB : Flutter est en phase de migration de **Material 2** vers **Material 3**. Néanmoins, cette migration n’étant pas terminée, nous vous recommandons de développer, durant ce cours, avec **Material 2**.
* L’app étend **StatelessWidget**, c’est donc un widget. En Flutter, presque tout est un **Widget**, incluant l’alignement, le padding et le layout.
* Le **Scaffold** widget, de la librairie **Material**, fournit une app bar par défaut, un titre, et un propriété de type **body** qui contient l’arbre de widgets pour l’écran « **Home** ».
* Le job principal d’un widget est de fournir une méthode **build** qui décrit comment afficher le widget en terme de widgets “enfants”.
* Le **body** consiste en un widget **Center** contenant un widget “enfant” de type **Text**. Le widget **Center** aligne son arbre de widgets au centre de l’écran.

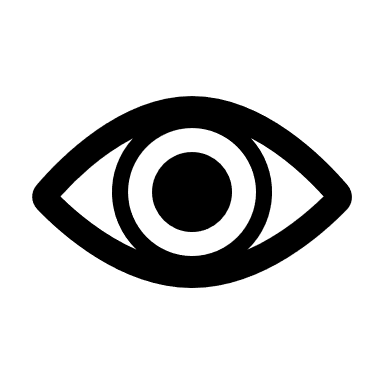
Une fois le code analysé et exécuté : **[commit** **avec message : FT01.2 app Flutter de base]**

## Modélisation possible du layout d’un écran

Veuillez lire le [Step 1](https://docs.flutter.dev/development/ui/layout/tutorial#step-1-diagram-the-layout) du tutoriel online.

## Création de layout et de stateless widgets de base

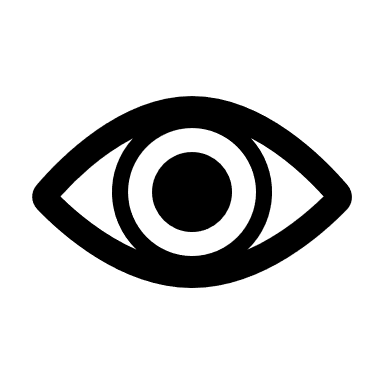
Veuillez réaliser le [Step 2](https://docs.flutter.dev/development/ui/layout/tutorial#step-2-implement-the-title-row) du tutoriel online.

****Observations & questions

* Il y a une méthode **build** dont le rôle est de construire la UI. Vous voyez qu’elle renvoie un objet de type **Widget**. Que représente un **Widget** ? Comment en construire ? Pour le savoir, visionnez **attentivement** la vidéo suivante : [How to create stateless widgets ?](https://www.youtube.com/watch?v=wE7khGHVkYY) (7 minutes, si vous avez du mal avec l’Anglais, mettez les sous-titres)
* Alternativement, si vous êtes plutôt lecteur que visionneur, vous pouvez consulter une des pages suivantes :
  + En Anglais : [introduction to widgets](https://flutter.dev/docs/development/ui/widgets-intro)
  + En Français, arrêtez-vous juste avant « Stateful Widget » : [Widget – State – Context - InheritedWidget](https://www.didierboelens.com/fr/2018/06/widget-state-context-inheritedwidget/)
* Qu’est-ce que la composition en Flutter ?
* A ce step 2, nous avons créé un **stateless widget** en tant que propriété de la méthode **build**. Comment s’appelle-t-il ?
* Les **stateless widgets** sont **immuables**, leur propriétés ne peuvent pas changer, toutes les valeurs sont « **final** ».
* Il existe un Flutter inspector permettant de visualiser et expérimenter avec vos layouts : regarder la vidéo [Layout Explorer](https://youtu.be/Jakrc3Tn_y4) (2 minutes), ou si vous préférez la lecture, vous pouvez lire [cet article](https://docs.flutter.dev/development/tools/devtools/inspector).

Une fois le code analysé et exécuté : **[commit** **avec message : FT01.3 title row]**

Veuillez maintenant réaliser le [Step 3](https://docs.flutter.dev/development/ui/layout/tutorial#step-3-implement-the-button-row) du tutoriel online.

****Observations & questions

* A ce step 3, nous avons créé une fonction qui permet de créer des **stateless widgets** contenant un bouton dans une colonne. Quel est l’intérêt de cette façon de faire ?

Une fois le code analysé et exécuté : **[commit** **avec message : FT01.4 button row]**

Veuillez maintenant réaliser le [Step 4](https://docs.flutter.dev/development/ui/layout/tutorial#step-4-implement-the-text-section) du tutoriel online.

Une fois le code analysé et exécuté : **[commit** **avec message : FT01.5 text section"]**

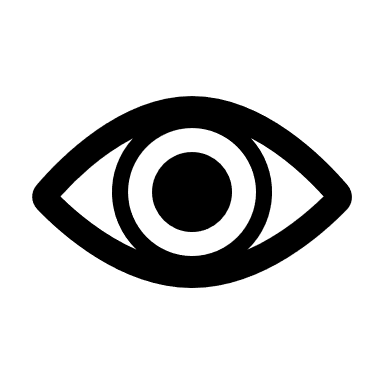
## Ajout d’une image dans votre app

Veuillez réaliser les [Step 5](https://docs.flutter.dev/development/ui/layout/tutorial#step-5-implement-the-image-section) & [Step 6](https://docs.flutter.dev/development/ui/layout/tutorial#step-6-final-touch) du tutoriel online.

Si votre image ne charge pas, faites attention à l’indentation dans **pubspec.yaml**, elle est importante :

flutter:  
  
 *# …* uses-material-design: true  
  
 assets:  
 - images/lake.jpg

Si ça ne charge toujours pas, lancer la commande **flutter clean** dans le terminal. Puis relancer votre application.

****Observations & questions

* Lorsque l’on ajoute des images à son application, ou tout autre asset, il est important de l’indiquer au niveau de **pubspec.yaml,** sinon le code ignore la disponibilité de vos assets.

Une fois le code analysé et exécuté : **[commit** **avec message : FT01.6 image section]**

## Création d’un stateless widget réutilisable & librairies

Nous allons temporairement nous éloigner du tutoriel online.

Le code de notre application crée une belle « carte » présentant un camping. Potentiellement, cette « carte » pourrait être reprise dans d’autres applications. Il serait donc utile de créer un widget réutilisable se trouvant dans son propre fichier et présentant une carte (sans le **Scaffold**…).

Veuillez créer un nouveau ficher dart dans le folder **lib**: **new -> Dart File**. Appelez le **my\_oeschinen\_card**.

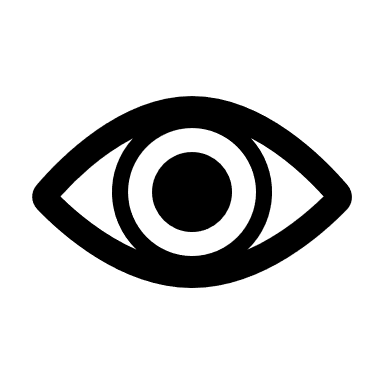
Dans ce fichier, créez un nouveau stateless widget en tapant **stless** dans votre IDE et en donnant le nom **MyOeschinenCard**

Faites l’import de la dépendance manquante en passant votre souris sur xWidgetx .

Suite à cela, votre fichier **my\_oeschinen\_card.dart** doit ressembler à ceci :

import 'package:flutter/material.dart';  
  
class MyOeschinenCard extends StatelessWidget {  
 const MyOeschinenCard({Key? key}) : super(key: key);  
  
 @override  
 Widget build(BuildContext context) {  
 Widget titleSection = Container(  
 padding: const EdgeInsets.all(32),  
 child: Row(  
 children: [  
 Expanded(  
 /\*1\*/  
 child: Column(  
 crossAxisAlignment: CrossAxisAlignment.start,  
 children: [  
 /\*2\*/  
 Container(  
 padding: const EdgeInsets.only(bottom: 8),  
 child: const Text(  
 'Oeschinen Lake Campground',  
 style: TextStyle(  
 fontWeight: FontWeight.*bold*,  
 ),  
 ),  
 ),  
 Text(  
 'Kandersteg, Switzerland',  
 style: TextStyle(  
 color: Colors.*grey*[500],  
 ),  
 ),  
 ],  
 ),  
 ),  
 /\*3\*/  
 Icon(  
 Icons.*star*,  
 color: Colors.*red*[500],  
 ),  
 const Text('41'),  
 ],  
 ),  
 );  
  
 Color color = Theme.*of*(context).primaryColor;  
  
 Column \_buildButtonColumn(Color color, IconData icon, String label) {  
 return Column(  
 mainAxisSize: MainAxisSize.min,  
 mainAxisAlignment: MainAxisAlignment.center,  
 children: [  
 Icon(icon, color: color),  
 Container(  
 margin: const EdgeInsets.only(top: 8),  
 child: Text(  
 label,  
 style: TextStyle(  
 fontSize: 12,  
 fontWeight: FontWeight.*w400*,  
 color: color,  
 ),  
 ),  
 ),  
 ],  
 );  
 }  
  
 Widget buttonSection = Row(  
 mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
 children: [  
 \_buildButtonColumn(color, Icons.*call*, 'CALL'),  
 \_buildButtonColumn(color, Icons.*near\_me*, 'ROUTE'),  
 \_buildButtonColumn(color, Icons.*share*, 'SHARE'),  
 ],  
 );  
  
 Widget textSection = const Padding(  
 padding: EdgeInsets.all(32),  
 child: Text(  
 'Lake Oeschinen lies at the foot of the Blüemlisalp in the Bernese '  
 'Alps. Situated 1,578 meters above sea level, it is one of the '  
 'larger Alpine Lakes. A gondola ride from Kandersteg, followed by a '  
 'half-hour walk through pastures and pine forest, leads you to the '  
 'lake, which warms to 20 degrees Celsius in the summer. Activities '  
 'enjoyed here include rowing, and riding the summer toboggan run.',  
 softWrap: true,  
 ),  
 );  
  
 return Column(children: [  
 Image.asset(  
 "images/lake.jpg",  
 width: 600,  
 height: 240,  
 fit: BoxFit.cover,  
 ),  
 titleSection,  
 buttonSection,  
 textSection  
 ]);  
 }  
}

Vous allez appeler votre nouveau widget réutilisable au sein de **main.dart** afin d’afficher deux fois la « carte » de présentation du camping du lac.

****Observations & questions

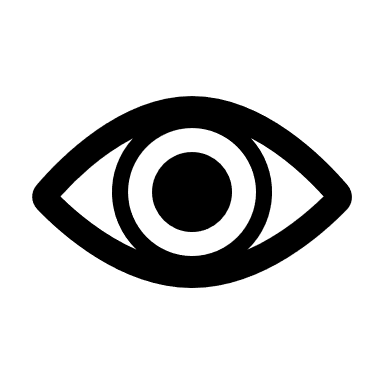
* Où faire l’appel du nouveau composant ? Tentez de le faire avant de lire la suite ; )

Vous devriez avoir ajouté l’appel à ce widget au sein de la **MaterialApp** renvoyée dans votre **main.dart** :

body: ListView(  
 children: [  
 Image.asset('images/lake.jpg',  
 width: 600, height: 240, fit: BoxFit.cover),  
 titleSection,  
 buttonSection,  
 textSection,  
 const MyOeschinenCard() // NEW TO PRINT A SECOND TIME THE SAME CARD

On renvoie une **ListView** à ce stade-ci car c’est un élément de layout qui offre un widget scrollable verticalement.

Veuillez veiller à ce que votre librairie **my\_oeschinen\_card.dart** soit importée : si nécessaire, faites l’import au passage de votre souris sur **MyOeschinenCard**.

****Observations & questions

* Tout comme en JS nous avons la possibilité de créer des modules ; en Dart, cela se fait via le concept de **library**. Il suffit de :
  + créer un fichier **file\_name.dart ;**
  + faire l’import de ce fichier dans le script où vous souhaitez utiliser des classes, objets, fonctions, variables ou constantes mises à disposition ;
  + rendre privé les éléments nécessaires en ajoutant un préfixe au nom de ces éléments : **\_identifier**
* Vous avez déjà rencontré une méthode qui était privée au sein du tutoriel. Cela vous revient ?
* **MyOeschinenCard** est un widget très spécifique. Il serait intéressant de créer un composant plus générique que l’on puisse paramétrer en lui passant l’URI d’une image, des titres, une description…

Une fois le code analysé et exécuté : **[commit** **avec message : FT01.7 création d’un stateless widget réutilisable"]**

## Création et paramétrisation d’un stateless widget

Afin d’améliorer notre application et rendre notre « carte » potentiellement utilisable dans d’autres contextes (présentation de restaurants, activités, lieux…), nous allons créer un **stateless widget** réutilisable, nommé **MyCard** au sein du fichier **my\_card.dart**.

Nous devons pouvoir facilement paramétrer le widget **MyCard** ; nous souhaitons :

* lui indiquer l’URI d’une image en pouvant nommer ce paramètre **imageUri** ; si celle-ci n’existe pas, il faut présenter l’image identifiée dans **pubspec.yaml** par **images/default.jpg**NB : nous vous proposons de télécharger et d’utiliser cette [image-ci](https://unsplash.com/photos/gnyA8vd3Otc).
* lui donner un titre de niveau 1 en pouvant nommer le paramètre **title1** ; **title1** ne peut pas être nul et n’a pas de valeur par défaut ;
* un titre de niveau 2 en nommant le paramètre **title2** ; ce paramètre est optionnel et peut être nul ;
* spécifier le texte descriptif associé à la carte en pouvant nommer celui-ci via **description** ; ce paramètre peut être nul ;
* NB : les boutons (CALL, ROUTE & SHARE), à ce stade-ci, ne doivent pas être paramétrables.

Veuillez ajouter le fichier **my\_card.dart** à votre projet ainsi que l’image par défaut (pour rappel, cette [image-ci](https://unsplash.com/photos/gnyA8vd3Otc)).

Pensez à informer à votre application de la nouvelle image en mettant à jour **pubspec.yaml** :

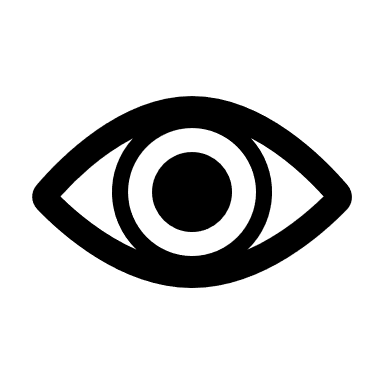
assets:  
 - images/lake.jpg  
 - images/default.jpg *# new code*

Voici le code de **my\_card.dart** :

import 'package:flutter/cupertino.dart';  
import 'package:flutter/material.dart';  
  
class MyCard extends StatelessWidget {  
 final String title1;  
 final String? title2;  
 final String imageUri;  
 final String? description;  
  
 static const int *DEFAULT\_LIKE\_COUNT* = 1;  
  
 const MyCard({Key? key, required this.title1, this.title2, this.imageUri = "images/default.jpg",  
 this.description })  
 : super(key: key);  
  
 @override  
 Widget build(BuildContext context) {  
 Widget titleSection = Container(  
 padding: const EdgeInsets.all(32),  
 child: Row(  
 children: [  
 Expanded(  
 /\*1\*/  
 child: Column(  
 crossAxisAlignment: CrossAxisAlignment.start,  
 children: [  
 /\*2\*/  
 Container(  
 padding: const EdgeInsets.only(bottom: 8),  
 child: Text(  
 title1,  
 style: const TextStyle(  
 fontWeight: FontWeight.*bold*,  
 ),  
 ),  
 ),  
 Text(  
 title2 ?? "",  
 style: TextStyle(  
 color: Colors.*grey*[500],  
 ),  
 ),  
 ],  
 ),  
 ),  
 /\*3\*/  
 Icon(  
 Icons.*star*,  
 color: Colors.*red*[500],  
 ),  
 Text(*DEFAULT\_LIKE\_COUNT*.toString()),  
 ],  
 ),  
 );  
  
 Color color = Theme  
 .*of*(context)  
 .primaryColor;  
  
 Widget buttonSection = Row(  
 mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
 children: [  
 \_buildButtonColumn(color, Icons.*call*, 'CALL'),  
 \_buildButtonColumn(color, Icons.*near\_me*, 'ROUTE'),  
 \_buildButtonColumn(color, Icons.*share*, 'SHARE'),  
 ],  
 );  
  
 Widget textSection = Padding(  
 padding: const EdgeInsets.all(32),  
 child: Text(  
 description ?? "",  
 softWrap: true,  
 ),  
 );  
  
 return Column(  
 children: [  
 Image.asset(imageUri,  
 width: 600, height: 240, fit: BoxFit.cover),  
 titleSection,  
 buttonSection,  
 textSection,  
 ],  
 );  
 }  
  
 Column \_buildButtonColumn(Color color, IconData icon, String label) {  
 return Column(  
 mainAxisSize: MainAxisSize.min,  
 mainAxisAlignment: MainAxisAlignment.center,  
 children: [  
 Icon(icon, color: color),  
 Container(  
 margin: const EdgeInsets.only(top: 8),  
 child: Text(  
 label,  
 style: TextStyle(  
 fontSize: 12,  
 fontWeight: FontWeight.*w400*,  
 color: color,  
 ),  
 ),  
 ),  
 ],  
 );  
 }  
  
  
}

Voici la MàJ de **main.dart** pour appeler deux fois ce widget :

body: ListView(  
 children: [  
 Image.asset('images/lake.jpg',  
 width: 600, height: 240, fit: BoxFit.cover),  
 titleSection,  
 buttonSection,  
 textSection,  
 const MyOeschinenCard(),  
 const MyCard(title1: "Title 1"), // New code  
 const MyCard(  
 imageUri: "images/lake.jpg",  
 title1: "Oeschinen Lake Camping",  
 title2: "CH",  
 description: "Wow... Nowhere is better than this...",  
 ), // New code  
 ],  
),

****Observations & questions

* L’image « default.jpg » ne s’affiche pas ? Veuillez stopper et démarrer votre application… votre asset devrait maintenant être chargé.
* Est-ce que vous comprenez bien cette importante ligne ?

const MyCard({Key? key, required this.title1, this.title2, this.imageUri = "images/default.jpg",  
this.description })  
 : super(key: key);

* Si la réponse est non, alors il est important de relire ces sections :
  + [Optional named parameters](https://dart.dev/codelabs/dart-cheatsheet#optional-named-parameters)
  + [Using this in a constructor](https://dart.dev/codelabs/dart-cheatsheet#using-this-in-a-constructor)

Une fois le code analysé et exécuté : **[commit** **avec message : FT01.8 paramétrisation d’un stateless widget"]**

NB : 🍬 Si vous souhaitez en apprendre d’avantage sur les stateless widget : [StatelessWidget class](https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html).

## Comprendre les contraintes

Félicitations, vous avez créé vos premiers layouts en Flutter 🎉 !

Néanmoins, ça n’est pas évident de comprendre comment les largeurs, hauteurs, alignements… sont gérés. Les layouts en Flutter sont très différents des layouts en HTML / CSS.

Mais plus vous allez pratiquer Flutter, plus cela va devenir naturel.

Voici une règle très importante : **les contraintes vont vers le bas, les tailles vers le haut et le parent définit la position.**

Voici, de manière plus détaillée, comment le layout des widgets est déterminé :

* Un widget reçoit ses 4 **contraintes** de son **parent**. : minimum & maximum width, minimum & maximum height.
* Ensuite le widget va indiquer à ses **enfants** leurs **contraintes**, puis il demande à chaque enfant quelle **taille** il veut être.
* Ensuite le widget **positionne** ses enfants, **horizontalement** selon **l’axe x**, **verticalement** selon **l’axe y**, un par un.
* Finalement, le widget dit à son **parent** sa **taille** en respectant les contraintes reçues.

Voici les limitations :

* Un widget peut donc décider de sa taille seulement si cette taille respecte les contraintes reçues du parents !
* Un widget ne peut pas décider de sa position à l’écran (c’est le parent qui le fait) !
* Il faut prendre tout l’arbre des widgets pour déterminer la position d’un widget en particulier (c’est pas possible de le faire de manière isolée)
* La taille d’un enfant peut être ignorée si un parent n’a pas suffisamment d’info comment aligner un enfant qui aurait une taille différente.

Pour bien comprendre comment sont gérés les contraintes de taille et de position, vous pouvez jeter un œil aux exemples donnés sur : [Understanding constraints](https://docs.flutter.dev/development/ui/layout/constraints#:~:text=A%20widget%20can%20decide%20its,the%20position%20of%20the%20widget).

# Exercice

## Introduction

Mais qui connait les prix Nobel ? Peut-être connaissez-vous le prix Nobel concernant la création de la dynamite… mais sinon, toutes ces sortes de « super héros » (scientifiques, artistiques, bienveillants…) sont bien souvent méconnus du public 😥.

Ils risquent donc de sombrer dans l’oubli…

Et c’est là que nous, créateurs d’apps, nous allons intervenir : remettons en lumière ces événements associés au prix Nobel.

Une API Open Source nous offre toutes les données nécessaires : <http://api.nobelprize.org/v1/prize.json>

**Veuillez créer un nouveau projet (New Flutter Project) nommé *ex1* dans votre repository de cours.**

## HomeScreen via un stateless widget

Nous souhaitons créer un **HomeScreen**, un stateless widget, qui affiche les données d’un lauréat d’un prix Nobel. Le **HomeScreen** doit contenir un **Scaffold** et un titre. Dans un premier temps, le code ne doit pas être trop structuré et toute l’UI se trouve dans le fichier **main.dart**.

Les données du lauréat doivent se trouver de cette **map** que vous ajouterez tout à la fin de votre fichier **main.dart** :

const laureate1 = {  
 "id": "819",  
 "firstname": "Al",  
 "surname": "Gore",  
 "motivation":  
 "\"for their efforts to build up and disseminate greater knowledge about man-made climate change, and to lay the foundations for the measures that are needed to counteract such change\"",  
 "share": "2"  
};

Veuillez-vous « nourrir » de cette map pour afficher la **motivation**, puis le **firstname** et **surname**.

Voilà à quoi pourrait ressembler votre application. **Ne passez pas trop de temps à peaufiner votre layout à cette étape-ci**!

Graphical user interface, text

Description automatically generated

**⚡ [commit** **avec message : F01.1 création du HomeScreen]** ⚡

## Création d’un stateless widget paramétrable

Dans l’optique d’une future application qui pourra afficher de nombreux lauréats, vous devez créer un stateless widget qui doit :

* être paramétrable et réutilisable. Il doit donc se trouver dans une **librairie** ;
* indiquer le **firstname**, **surname** du lauréat ainsi que la **motivation** de lui avoir donné le prix Nobel ; ces informations doivent être données par des arguments **nommés**, ne pouvant pas être nuls :
  + **firstname** est optionnel, sa valeur par défaut est "" ;
  + **surname** est optionnel, sa valeur par défaut est "" ;
  + **motivation** est obligatoire.

Votre **HomeScreen**  doit appeler votre nouveau widget présentant un lauréat.

Quand tout est fonctionnel comme au point précédent (fin du §7.2) :

**⚡ [commit avec message : F01.2 création d’un widget paramétrable dans une librairie]** ⚡

## Peaufinage du layout

Veuillez changer le layout de votre application. Vous devez :

* Jouer avec la couleur d’un des textes.
* Ajouter de l’espace entre le titre du **HomeScreen** et l’affichage du lauréat.

Voilà à quoi pourrait ressembler votre application.

Graphical user interface, text, application, chat or text message

Description automatically generated

Si vous essayez d’avoir un layout qui correspond à l’écran proposé, cela risque de vous prendre beaucoup de temps. N’hésitez pas à simplifier le layout, en allant au plus simple.

Besoin d’inspiration pour les layouts ? [layout widgets](https://docs.flutter.dev/development/ui/widgets/layout) et [Components](https://m2.material.io/components?platform=flutter).

**[commit avec message : F01.3 stateless widget peaufiné]**

## 🍬 Challenge optionnel

Reprenez le tutorial et tentez de peaufiner le layout pour que chaque image prenne toute la largeur de l’écran.

Vous pourriez donc mettre à jour **MyCard** & **MyOeschinenCard** pour avoir ce genre de résultat :

Graphical user interface, website

Description automatically generated