

# FLUTTER SUPPORT

---

COURS L2 & L3 2022 CODING FACTORY



# ORIGINE DE FLUTTER

---

- Mai 2017
- Open Source
- Android, iOS, Linux, Mac, Windows, Google Fuchsia et le web à partir d'une seule base de code
- Responsable : Google

# TECHNOLOGIES

---

- **Flutter est** un kit de développement d'interface utilisateur open-source
- Repose sur le langage de programmation DART
- Documentation : <https://api.flutter.dev/>
- Version de Flutter actuelle : 3.3.3
- Moteur codé en C/C++

# AVANTAGES DE L'ALLIANCE DART/FLUTTER

---

- Dart constitue également un atout majeur de Flutter. En effet, ce langage peut supporter les deux types de compilations JIT (Just-in-Time) et AOT (Ahead-of-Time) .
- Pendant la phase de développement, la compilation JIT est utilisée afin de permettre le hot reload grâce à DartVM. Flutter injecte alors instantanément les changements sans reconstruire toute l'application, ce qui optimise significativement les temps de développement.
- Lors de la phase de déploiement sur les stores, Flutter utilise la compilation AOT afin de générer une application 100 % native sans aucun compromis pour les performances.

# INSTALLATION DE FLUTTER

---

- Doc officiel Multi OS
- <https://docs.flutter.dev/get-started/install>
- En vidéo (conseillé):
- <https://www.youtube.com/watch?v=yIQMmj5UuJA>
- En cas de problème de PATH :
- `export PATH=[PATH_TO_FLUTTER_GIT_DIRECTORY]/flutter/bin:$PATH`



# COMMANDES IMPORTANTE

---

Une fois installé Flutter dispose de plusieurs commandes à garder en tête à exécuter dans un shell :

- flutter doctor
  - Permet de vérifier l'état de votre installation
- flutter upgrade
  - Permet de mettre à jour une installation
- flutter create <nomApp>
  - Pour créer une application Flutter

# COMMANDES IMPORTANTE

---

- flutter devices -d <DEVICE\_ID>
  - Permet de lister les devices connecté
- flutter emulators
  - Lister, lancer créer de nouveaux émulateurs
- Liste complète des commandes :
  - <https://docs.flutter.dev/reference/flutter-cli>

# NOUVEAU PROJET AVEC ANDROID STUDIO

**New Project**

Project name: L2CERGY

Project location: C:\Users\Elodie\StudioProjects\L2CERGY

Description: A new Flutter project.

Project type: Application

Organization: fr.codingfactory

Android language: ☐ Java ☒ Kotlin

iOS language: ☐ Objective-C ☒ Swift

Platforms: ☒ Android ☒ iOS ☐ Linux ☐ MacOS ☒ Web ☒ Windows

Platform availability might depend on your Flutter SDK channel, and which desktop platforms have been enabled.

Additional desktop platforms can be enabled by, for example, running "flutter config --enable-linux-desktop" on the command line.

When created, the new project will run on the selected platforms (others can be added later).

☒ Create project offline

**More Settings**

Module name: L2CERGY

Content root: C:\Users\Elodie\StudioProjects\L2CERGY

Module file location: C:\Users\Elodie\StudioProjects\L2CERGY

Project format: .idea (directory based)

Previous Finish Cancel Help



# IMPORT DE BASE

---

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

Pour construire vos interfaces ils vous faudra obligatoirement avoir cet import en entête de classe.

## Autres librairies

Animation :: The Flutter animation system.

Cupertino :: Flutter widgets implementing the current iOS design language.

Foundation :: Core Flutter framework primitives.

Gestures :: The Flutter gesture recognizers.

Material :: Flutter widgets implementing Material Design.  
Painting :: The Flutter painting library.  
Physics :: Simple one-dimensional physics simulations, such as springs, friction, and gravity, for use in user interface animations.  
Rendering :: The Flutter rendering tree.  
Scheduler :: The Flutter Scheduler library.  
Semantics :: The Flutter semantics package.  
Services :: Platform services exposed to Flutter apps.  
Widgets :: The Flutter widgets framework.

# DIFFÉRENCE ENTRE STATELESS WIDGET ET STATE FULL WIDGET

---

- Voir ce site qui donne une très bonne explication de cette différence.
- <https://neptuneapps.com/difference-entre-stateless-widget-et-stateful-widget/>

# LES FUTURES ET LES ISOLATES (FONCTIONS SYNCHRONES / ASYNC

---

- <https://www.didierboelens.com/fr/2019/01/futures-isolates-event-loop/>

# AJOUTER DES PAGES À SON APPLI

---

```
class SecondPage extends StatefulWidget {  
  static const tag = "second_page";  
  @override  
  State<StatefulWidget> createState() => _MySecondPageState();  
}
```

- Dans votre classe qui contient le Material.app ajouté cette propriété :

```
routes: {SecondPage.tag : (context)=>SecondPage()},
```

- Sur le bouton qui doit lancer la seconde page :

```
onPressed: (){Navigator.of(context).pushNamed(SecondPage.tag);},
```

# LA CRÉATION DES OBJETS EN DART

---

- <https://pythonforge.com/dart-classes-heritage/>



# STOCKAGE / BDD NOSQL

---

- Avec Firebase
- <https://www.youtube.com/watch?v=IPMlcGTzxGc>
- Avec MongoDB
- [https://www.youtube.com/playlist?list=PLSuzwxF6LC4BI Z2fylAI yTqPfrnmrTKW\\_](https://www.youtube.com/playlist?list=PLSuzwxF6LC4BI Z2fylAI yTqPfrnmrTKW_)

# ECHANGER DES DONNÉES ENTRE PAGES

---

- Deux possibilités
  - la première passer à la seconde page via le Navigator une collection devant être affiché dans un list/gridView de la page d'appel et rafraichir la première page une fois que la seconde n'est plus affiché.
  - La seconde :Appeler de manière asynchrone une autre page qui nous retournera une valeur également via le navigator.

# ECHANGER DES DONNÉES ENTRE PAGES

---

- Première solution :
  - Page Home depuis le bouton : ou users est la collection que l'ont passe en argument
    - `onPressed: () {Navigator.of(context).pushNamed(SecondPage.tag, arguments: users).then((_) => setState(() {}))};`
- Page 2
  - Depuis le widget Build
    - `List<Person> users = ModalRoute.of(context)!.settings.arguments as List<Person>;`
  - depuis le bouton de validation du formulaire
    - `users.add(Person).`

# ECHANGER DES DONNÉES ENTRE PAGES

---

- Seconde solution :
- Page 1 :
  - à partir d'un bouton => onPressed: (){\_awaitReturnValueFromSecondScreen(context);},
  - Création Async Méthode :

```
void _awaitReturnValueFromSecondScreen(BuildContext context) async {  
  // start the SecondScreen and wait for it to finish with a result  
  final result = await Navigator.push(  
    context,  
    MaterialPageRoute(  
      builder: (context) => const SecondPage(),  
    ));  
  
  // after the SecondScreen result comes back update the Text widget with it  
  setState(() {  
    cards.add(result);  
  });  
}
```

- Page 2 : sur le bouton qui valide le formulaire : Navigator.pop(context, card);

# INSTANCIER UN OBJET À PARTIR D'UNE COLLECTION MONGODB

---

- Après avoir récupérer la collection de notre choix on appelle une méthode Future<> async:

```
var users = await collection?.find().toList();           // Retour de la collection issu de Mongo qu'on transforme en liste (format JSON)
```

```
List<User> usersList = [];                               // Liste qui va stocker tous les users de la collection
```

```
users?.forEach((item) {
```

```
    final user = User(item['nom'], item['prenom'], item['pwd'], item['ffelink']);
```

```
    usersList.add(user);
```

```
}
```

```
return usersList;
```



# INSTANCIER UN OBJET À PARTIR D'UNE COLLECTION MONGODB - AFFICHAGE

---

- Après avoir récupérer et instancier des objets Users à partir de la requête sur MongoDB dans mon scaffold je dois utiliser un widget qui permet de tous les afficher dans une optique Future :

FutureBuilder(

future: MongoDBBase getUsers(), // Appel à la fonction de la slide précédente qui va nous retourner une Liste<User>

builder: (context, AsyncSnapshot snapshot) {

if (snapshot.connectionState == ConnectionState.waiting) { // Si nous n'avons pas encore eu la réponse de la BDD nous affichons une barre de progress circulaire

return const Center(child: CircularProgressIndicator(), );

} else {

if (snapshot.hasData) { // Si l'appel à notre fonction Future retourne des éléments

var totalData = snapshot.data.length; // Savoir le nombre de USERS et donc le nombre de fois que devra itérer notre boucle (GridView.Builder)

return GridView.builder(gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(crossAxisCount: 2, ), // La grid aura deux colonnes

itemBuilder: (context, index) => displayUser(snapshot.data[index]), )}, // displayUser est une fonction qui retourne une Card