TP N°1 – Installation complète de Flutter et création d'une première application

Module : Développement Mobile Multiplatforme / ENIAD

Enseignant : Pr. Mohamed BOUDCHICHE

Année universitaire : 2025–2026

Objectif

Ce TP accompagne l'étudiant dans l'installation complète de l'environnement Flutter sur Windows, macOS ou Linux. À la fin de ce travail, vous serez capable de configurer Flutter, exécuter votre première application et comprendre la structure d'un projet Flutter.

1 Pré-requis

- Connexion Internet stable ;
- 10 Go d'espace libre minimum ;
- Droits d'administrateur sur la machine ;
- Ordinateur Windows 10/11, macOS ou Linux.

2 Installation des outils de base

2.1 Étape 1 – Installer Git

1. Rendez-vous sur `https://git-scm.com/downloads`.
2. Choisissez votre système :
   - Windows : exécutez le fichier `Git-setup.exe`.
   - macOS : installez via `brew install git`.
   - Linux : installez via `sudo apt install git`.
3. Acceptez les paramètres par défaut (`Next → Next → Finish`).

Commande à exécuter

```
git --version
```

Commande à exécuter

```
git config --global user.name "Votre Nom"
git config --global user.email "votre.email@example.com"
```

Remarque importante

Git est indispensable : Flutter utilise des dépôts Git pour télécharger le SDK et les dépendances.

2.2 Étape 2 – Installer Visual Studio Code

1. Téléchargez depuis `https://code.visualstudio.com/`.
2. Installez normalement et ouvrez VS Code.
3. Ajoutez les extensions :
   — **Flutter**
   — **Dart**
4. Vérifiez dans la barre d'état que le SDK Flutter est reconnu.

Astuce pratique

VS Code est léger et idéal pour le développement Flutter. Ouvrez la palette de commandes : `Ctrl + Shift + P → Flutter: Doctor` pour vérifier la configuration.

3 Installation du SDK Flutter

3.1 Étape 3 – Téléchargement manuel du SDK

1. Accédez à `https://flutter.dev/docs/get-started/install`.
2. Téléchargez le SDK selon votre système :
   — Windows : archive ZIP.
   — macOS : archive .zip (Intel ou Apple Silicon).
   — Linux : archive .tar.xz.
3. Extrayez-le dans le dossier : `C:\src\flutter`
4. Sous Linux :

Commande à exécuter

```
tar xf flutter_linux.tar.xz
```

Remarque importante

Choisissez un dossier simple sans espace dans le nom (évitez "Program Files").

3.2 Étape 4 – Ajouter Flutter au PATH

— Windows : Ouvrez les variables d'environnement → Path → Nouveau → `C:\src\flutter\bin`.
— macOS / Linux :

Commande à exécuter

```
export PATH="$PATH:/home/utilisateur/flutter/bin"
```

Commande à exécuter

```
flutter --version
```

Erreur fréquente

Si la commande `flutter` n'est pas reconnue, redémarrez le terminal ou vérifiez le PATH.

4 Installation d'Android Studio et du SDK Android

1. Téléchargez Android Studio : `https://developer.android.com/studio`.
2. Installez avec les options par défaut.
3. Ouvrez Android Studio → `File → Settings → Plugins` → installez **Flutter** et **Dart**.
4. Configurez le SDK Android :
   - `File → Settings → Appearance → System Settings → Android SDK`.
   - Onglet **SDK Tools** → cochez **Android SDK Command-line Tools (latest)**.

Commande à exécuter

```
flutter doctor --android-licenses
```

Astuce pratique

Acceptez toutes les licences Android affichées pour permettre la compilation d'APK.

5 Vérification avec Flutter Doctor

Commande à exécuter

```
flutter doctor -v
```

Remarque importante

Chaque ligne avec un `[√]` indique une installation réussie. Les lignes avec `[X]` signalent un problème à corriger.

Erreur fréquente

Problème courant : `cmdline-tools component is missing`. → Ouvrez Android Studio → SDK Manager → SDK Tools → cochez "Command-line Tools".

6 Création d'un projet Flutter

1. Ouvrez VS Code → `Ctrl + Shift + P → Flutter: New Project`.
2. Sélectionnez **Application Flutter**.
3. Donnez un nom : `flutter_app`.
4. Attendez la création du projet.

5. Exécutez :

Commande à exécuter

```
flutter run -d chrome
```

6.1 Structure du projet Flutter

Lorsqu'un nouveau projet Flutter est créé, une arborescence complète de fichiers et de dossiers est générée automatiquement. Chaque élément joue un rôle bien défini dans le fonctionnement global de l'application.

— **lib/** : C'est le répertoire principal de votre code source en langage `Dart`. Il contient notamment le fichier `main.dart`, qui représente le **point d'entrée** de l'application Flutter. C'est dans ce fichier que la fonction principale `main()` appelle `runApp()` pour exécuter votre application. Vous pouvez y organiser votre code en plusieurs fichiers ou sous-dossiers (ex. `lib/screens`, `lib/widgets`, `lib/models`) pour séparer les fonctionnalités et rendre le projet plus clair.

Astuce pratique

Dans les projets de grande taille, il est recommandé de structurer le dossier `lib/` par modules : par exemple, créer un dossier `pages/` pour les interfaces et un dossier `services/` pour les fonctions logiques.

— **android/** et **ios/** : Ces deux dossiers contiennent le code natif spécifique à chaque plateforme. Flutter se charge de compiler le code Dart en code natif Android ou iOS via son moteur. Vous n'aurez généralement pas besoin de modifier ces fichiers sauf si vous souhaitez ajouter des permissions, modifier le nom de l'application, ou intégrer des librairies natives.

Remarque importante

Pour changer le nom de l'application ou l'icône, il faut modifier les fichiers situés dans ces répertoires (ex. `AndroidManifest.xml` pour Android, ou `Info.plist` pour iOS).

— **web/** : Présent si vous avez activé le support web. Il contient les fichiers nécessaires pour lancer votre application Flutter dans un navigateur (HTML, JavaScript, CSS).

— **test/** : Dossier contenant des fichiers de tests unitaires pour vérifier le bon fonctionnement des fonctions et widgets. Par défaut, Flutter crée un fichier `widget_test.dart` servant d'exemple.

— **pubspec.yaml** : Fichier de configuration central du projet Flutter. Il indique :

  — le nom du projet et sa version ;
  — la version minimale de Flutter requise ;
  — la liste des dépendances (packages) utilisées dans le projet ;
  — les ressources (images, polices, sons, etc.) intégrées à l'application.

  C'est un fichier essentiel que Flutter lit à chaque compilation pour savoir quels modules importer et quelles ressources charger.

Astuce pratique

Après toute modification du fichier `pubspec.yaml`, exécutez la commande : `flutter pub get` Cela télécharge et met à jour les dépendances déclarées.

— **build/** : Ce dossier est généré automatiquement lors de la compilation. Il contient les fichiers intermédiaires et binaires (APK, fichiers web compilés, etc.). Vous pouvez le supprimer sans risque ; Flutter le régénérera au prochain build.

— **.dart_tool/** et **.idea/** : Ce sont des dossiers cachés utilisés respectivement par Flutter/Dart et par votre IDE (VS Code ou Android Studio). Ils stockent des informations de configuration locales du projet.

Remarque importante

Les fichiers et dossiers précédés d'un point (comme `.dart_tool` ou `.gitignore`) sont cachés dans le système, car ils ne doivent pas être modifiés manuellement. Ils servent à la gestion interne du projet et des dépendances.

6.2 Code de base

```
import 'package:flutter/material.dart';
void main() => runApp(MyApp());
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Hello Flutter',
      home: Scaffold(
        appBar: AppBar(title: Text('Ma première app Flutter')),
        body: Center(child: Text('Bienvenue à l'ENIAD !')),
      ),
    );
  }
}
```

Astuce pratique

Modifiez le texte et testez le **Hot Reload** : tapez `r` dans le terminal pour voir les changements instantanément.

7 Exercices de validation

Les exercices suivants permettent de vérifier que votre environnement Flutter fonctionne correctement, aussi bien sur le Web que sur un appareil ou émulateur Android.

1. Modifier le texte affiché dans `main.dart`
   Exemple : remplacez le message par : `"Bonjour Flutter depuis l'ENIAD !"`
2. Changer la couleur principale de l'application.
   Dans le code, modifiez :

```
theme: ThemeData(primarySwatch: Colors.blue)
```

en choisissant une autre couleur (ex. `Colors.green`, `Colors.orange`, etc.)

3. Ajouter un bouton qui affiche un message à l'écran :

```
ElevatedButton(
  onPressed: () {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Bouton cliqué !')),
    );
  },
  child: Text('Appuyer ici'),
)
```

4. Exécuter et tester le projet sur différentes plateformes :
   - **Exécution sur le Web (Google Chrome) :**

     Commande à exécuter

     flutter run -d chrome

     Cette commande ouvre l'application Flutter directement dans le navigateur.
   - **Exécution sur un émulateur Android :**
     a) Ouvrez Android Studio → `Tools → Device Manager`.
     b) Créez un nouvel appareil virtuel (ex. Pixel 5, API 34).
     c) Démarrez l'émulateur.
     d) Vérifiez que Flutter le détecte :

        Commande à exécuter

        flutter devices

     e) Lancez l'application :

        Commande à exécuter

        flutter run -d emulator-5554

        (le numéro de l'émulateur peut varier selon votre machine)
   - **Exécution sur un téléphone Android réel (optionnel) :**
     a) Activez le *Mode développeur* sur le téléphone.
     b) Activez le *Débogage USB*.
     c) Connectez le téléphone via câble USB.
     d) Vérifiez qu'il est détecté :

        Commande à exécuter

        flutter devices

e) Exécutez :

Commande à exécuter

```
flutter run -d <nom_du_téléphone>
```

Astuce pratique

Vous pouvez basculer entre les plateformes de test avec le paramètre `-d` : `flutter run -d chrome`, `flutter run -d windows`, ou `flutter run -d emulator-5554`.

Remarque importante

L'exécution sur le Web permet une vérification rapide de l'interface, tandis que l'exécution sur Android valide toute la chaîne de compilation (SDK, build, permissions, etc.).

8 Annexe – Liens utiles

— Flutter SDK : `https://flutter.dev/docs/get-started/install`
— VS Code : `https://code.visualstudio.com/`
— Android Studio : `https://developer.android.com/studio`
— Git : `https://git-scm.com/downloads`

9 Checklist de vérification

☐ Git installé et configuré
☐ VS Code installé avec extensions Flutter/Dart
☐ Flutter SDK reconnu
☐ Android Studio et SDK Tools installés
☐ Commande `flutter doctor` sans erreurs
☐ Application Flutter exécutée avec succès

Fin du TP N°1 – Vous avez installé et configuré Flutter avec succès.■