Fiche procédure Flutter

Table des matières

Installation de Flutter

1. Télécharge le SDK Flutter
   Va sur flutter.dev et clique sur "Get started".
   Sélectionne ton système d’exploitation (Windows, macOS, Linux) et suis les instructions pour télécharger et extraire le SDK Flutter sur ton ordinateur.
2. Ajoute Flutter à ton PATH
   Ajoute le dossier `flutter/bin` à la variable d’environnement PATH pour utiliser la commande `flutter` dans le terminal.
3. Installe un éditeur de code
   Installe Visual Studio Code ou Android Studio, deux éditeurs très utilisés pour Flutter.
4. Installe les extensions Flutter et Dart
   Dans ton éditeur, ajoute les extensions Flutter et Dart pour bénéficier de l’autocomplétion, du débogage et d’autres outils utiles.
5. Vérifie l’installation
   Ouvre un terminal et tape :
6. `bash`

```
flutter doctor
```

7.
8. Suis les recommandations pour compléter l’installation (Android SDK, simulateurs, etc.).

Création d’un projet Flutter

1. Ouvre un terminal ou l’éditeur de code.
2. Crée un nouveau projet avec :
3. `bash`

```
flutter create mon_premier_projet
cd mon_premier_projet
```

4.

5. Lance l'application sur un émulateur ou un appareil connecté :
6. `bash`

```
flutter run
```

7.

8. *(Assure-toi d'avoir un simulateur Android/iOS ou un appareil branché).*

Utilisation de Flutter

- Structure d'un projet Flutter :
  - `lib/main.dart` : point d'entrée de l'application (fichier principal en Dart)
  - `pubspec.yaml` : gestion des dépendances et ressources
  - `android/`, `ios/`, `web/`, `windows/`, `linux/`, `macos/` : dossiers spécifiques aux plateformes
- Principe : Flutter fonctionne avec des widgets (éléments d'interface réutilisables) écrits en Dart.
- Hot reload : Permet de voir instantanément les modifications sans relancer l'application.

Bonnes pratiques

- Organise ton code en dossiers :
  `models/` (modèles de données), `screens/` (écrans), `widgets/` (composants réutilisables), `services/` (logique métier), `assets/` (ressources).
- Utilise des widgets réutilisables pour éviter la duplication de code.
- Gère l'état de l'application avec des solutions adaptées (Provider, Riverpod, Bloc…).
- Commente et documente ton code pour faciliter la maintenance.
- Teste régulièrement avec `flutter analyze` et mets en place des tests unitaires et d'intégration.
- Optimise les performances en évitant les widgets inutiles et en utilisant le hot reload.
- Centralise la gestion des thèmes pour garantir une cohérence visuelle.
- Utilise des dépendances fiables depuis pub.dev et limite leur nombre.
- Met en place un pipeline CI/CD pour automatiser les tests et déploiements.

Exemple de code Flutter

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

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Bienvenue sur Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Accueil Flutter'),
        ),
        body: const Center(
          child: Text(
            'Hello, Flutter!',
            style: TextStyle(fontSize: 28),
          ),
        ),
      ),
    );
  }
}
```